RESOURCE MANAGEMENT FOR TARGET TRACKING IN WIRELESS SENSOR NETWORKS

HAN MINGDING

NATIONAL UNIVERSITY OF SINGAPORE

2010

RESOURCE MANAGEMENT FOR TARGET TRACKING IN WIRELESS SENSOR NETWORKS

HAN MINGDING
(*B.Eng. (Hons), NUS*)

A THESIS SUBMITTED
FOR THE DEGREE OF MASTER OF ENGINEERING
DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING
NATIONAL UNIVERSITY OF SINGAPORE
2010

Abstract

Target tracking applications are popular in wireless sensor networks, in which distributed low-power devices perform sensing, processing and wireless communication tasks, for applications such as indoor localization with ambient sensors. Being resource-constrained in nature, wireless sensor networks require efficient resource management to select the most suitable nodes for sensing, in-network data fusion, and multi-hop data routing to a base-station, in order to fulfill multiple, possibly conflicting, performance objectives. For example, in target tracking applications, reducing sensing and update intervals to conserve energy could lead to a decline in application performance, in the form of tracking accuracy. In this thesis, we study resource management approaches to address such challenges, through simulations and test-bed implementations.

There are two main components of this thesis. We first address indoor target tracking using a state estimation algorithm and an information-driven sensor selection scheme. An information-utility metric is used to characterize application performance for adaptive sensor selection. We address the system design choices such as the system architecture and models, hardware, software and algorithms. We also describe the system implementation in a test-bed, which incorporates mobile devices such as smartphones, for control and monitoring of the wireless sensor network, querying of sensors, and visualization interfaces.

The second component is a simulation study of a distributed sensor election and routing scheme for target tracking in a multi-hop wireless sensor network. An objective function, which trades-off information-quality with remaining energy of nodes, is used for sensor election. Subsequently, energy-efficient multi-hop routing is performed back to the sink node. In our non-myopic approach, we convert the remaining energy of nodes into an additive cost-based metric, and next-hop nodes are selected based on the expected sum of costs to the base station. A decision-theoretic framework is formulated to capture the non-myopic decision-making problem, and a reinforcement learning approach is used to incrementally learn which nodes to forward packets to, so as to increase the delivery ratio at the sink node.

Acknowledgment

I would like to thank my supervisor Associate Professor Tham Chen Khong for his supervision and encouragement throughout my course of study.

Special thanks goes to Dr Lee-ling Sharon Ong of the National University of Singapore and Dr Wendong Xiao of the Institute for Infocomm Research for their help with the state estimation algorithms for filtering and sensor selection, as well as the test-bed implementations.

My thanks also go out to my friends who have encouraged and supported me through the course of my work, and most importantly, my family for their never-ending support.

September 19, 2010

Contents

List of Figures

Chapter 1

Introduction

This thesis addresses resource management approaches for target tracking applications in wireless sensor networks, by considering application-level performance such as tracking accuracy, and energy-efficient operation in order to increase network lifetime. A filtering approach is adopted for state estimation, and candidate sensors are selected based on information gain and remaining energy levels. Subsequently, the updated state estimate is forwarded to a sink node via multi-hop routing. A decision-theoretic approach is used for non-myopic decision-making by considering the expected sum of costs to the sink node.

Target tracking continues to be a popular application domain in wireless sensor networks. Besides outdoor tracking in unknown and harsh environments for military scenarios, target tracking has also been applied to indoor localization, such as in [1], which caters to the growing need for indoor human activity monitoring for elderly healthcare applications [2], and increasing interest in developing pervasive computing applications for smart-space environments [3]. While target tracking applications are used as a canonical example, the information-driven and energy-efficient approaches described can also be extended to other data-centric application domains in wireless sensor networks.

1.1 Resource Management in Wireless Sensor Networks

Wireless Sensor Networks (WSNs) consist of large numbers of low-power nodes, each with sensing, processing and wireless communication capabilities. While each node may lack resources for performing high-resolution sensing and fast computation, WSNs make use of sensor collaboration and in-network processing to overcome their resource limitations, and to provide redundancy to be robust to node failure [4]. The sensor coverage affects the ability of the application to respond quickly to local events while the rest of the WSN lies dormant in sleep mode, and nodes near the event-of-interest can collaborate to reduce redundant information. Sensor collaboration improves the confidence of sensing and estimation, filters out sensing noise, and reduces the amount of data communicated towards the sink node.

Being resource-constrained in nature, wireless sensor networks require efficient resource management to select the most suitable nodes for sensing, in-network data fusion, and data routing to a base-station node. Multiple performance objectives need to be fulfilled, which may conflict with one another. For example, in target tracking applications, reducing sensing and update intervals to conserve energy and prolong network lifetime could lead to a decline in application performance, such as tracking accuracy. In this thesis, we study resource management approaches to address such challenges, through simulations and test-bed implementations.

1.2 Sensor Data Fusion

In target tracking applications, estimation algorithms are used to keep track of detected targets, and sensors update the state estimates with their observations. However, the sensor observations may be noisy, so signal processing approaches are incorporated to filter out process and observation noise, and to incorporate readings from sensors. Data fusion combines signal processing with data aggregation, and information-driven sensor management approaches are desirable, where the information gain of a candidate sensor's observation is based on the current state estimate, and can be quantified as a utility metric. Information-theoretic measures such as entropy [5],[6], and divergence measures from estimation filters [7],[8] are some examples.

1.3 Distributed in-network Processing

In order to distribute the in-network processing across nodes, one approach is to address how to perform data and decision fusion [9] to trade-off communication and processing loads across sensors. Clustering mechanisms can be adopted, where cluster heads are chosen based on remaining energy levels. In heterogeneous node deployments, nodes with more processing and communication resources, such as faster processor speeds, more memory or higher bandwidth, can be chosen to be cluster head nodes.

Task scheduling approaches have also been adopted in WSNs, in which processing tasks can be modelled as a directed acyclic graph, and allocated to nodes to perform distributed processing, while constrained by a shared communication channel. Task scheduling can be performed for load balancing across nodes [10], subject to constraints on the schedule makespan. Due to the large solution space from large node deployments, as well as the computational complexity of scheduling algorithms, heuristic approaches are most commonly used used [11],[12]. A reinforcement learning approach was presented in [13], in which nodes learn which tasks to choose for a target tracking application.

1.4 Energy-Efficient Sensor Scheduling and Communication

Since wireless communication poses the most significant source of energy consumption in WSNs, there has been extensive research on designing energy-efficient wireless sensor networking protocols. Sleep-wake scheduling approaches focus on designing schedules for which a subset of nodes intermittently wakes up to maintain network connectivity and perform coarse-grained sensing to detect any events-of-interest, while the majority of the WSN lies dormant in a low-power sleep mode. Several schemes also look at transmission power control to adjust the communication range and network topology based on remaining energy of nodes, so as to reduce energy consumption and increase network lifetime.

At the wireless medium access control layer, energy-efficient MAC protocols have been proposed, such as long-preamble listening in B-MAC[14], synchronized duty-cycling in S-MAC [15], as well as carrier sensing approaches such as [16]. A component-based software architecture was presented in [17] for the design, implementation and evaluation of various energy-efficient MAC protocols.

1.5 Multi-hop Routing

In wireless sensor networks deployed in large geographic areas, the limited communication range of nodes, and the objective to conserve communication energy, makes it necessary to efficiently communicate data across multiple hops, from sensors that detect the events-of-interest to base station nodes. In contrast to routing protocols in mobile ad-hoc networks, wireless sensor network nodes are usually static, and energy-efficient and data-centric operation is desired, in addition to optimising network performance metrics such as delay and throughput.

Routing protocols also need to address frequent topology changes due to sleep-wake cycles, link and node failures. Routing protocols that focus on minimizing the sum of communication energy across nodes may result in some depleted nodes and unfair sensor utilisation along popular multi-hop paths. On the other hand, maximum lifetime routing provides a network-wide perspective, in which the network lifetime may be defined as the time till which the network first becomes partitioned. A comprehensive survey of the various challenges in wireless sensor networks from the data routing perspective is provided in [18].

Because of the possibly large numbers of deployed sensors, and the need for the ad-hoc network deployment to be self-organizing, node addressing schemes may not be feasible as they would incur high overhead. In many applications, getting the data about the sensed event-of-interest is often more important than the node identities, so a data-centric approach to sensor management is preferred over an address-centric approach. Due to high-density node deployments, multiple sensor nodes may detect the event-of-interest, so sensor collaboration is required to aggregate the sensed data so as to reduce transmissions and conserve energy. Routing of sensor queries and state information may also make use of information-based gradients, as presented in [19].

In [20], data is represented in attribute-value pairs, and nodes set up interests and information gradients between event and sink, so as to support ad-hoc querying, in-network caching of interests, and data aggregation. In [21], a family of negotiation-based protocols is presented, in which nodes advertise themselves when they receive updated information and subsequently, other nodes which are interested in the data request for it.

1.6 Decision-theoretic and Learning Approaches

Due to the various sources of uncertainty in wireless sensor networks, such as node failure and packet loss, estimation algorithms and communication protocols need to be able to incorporate probabilistic models of the target and network states. In addition, greedy solution approaches may not suffice, as a next-hop node may be chosen for its high remaining energy, but future hops towards the destination node may be depleted. Incorporating a longer decision-making horizon to maximise the sum of expected future rewards would provide better resource utilization and application performance in the longer-term.

However, decision-making with multi-step look-ahead often results in exponentially increasing computational time and space complexity, in order to seek an optimal decision among the entire state and action space over multiple steps. Optimal computation by dynamic programming is not feasible for resource-constrained sensor nodes.

Instead, learning-based approaches using a reward signal from the sensor network would be more suitable, as nodes are able to learn the immediate rewards from their actions, while they seek to maximise their long-term expected sum of rewards through trial-and-error. In addition, modeling and computational complexities have a much less significant effect and nodes can learn good *sample paths* as they explore the solution space. Here, it is assumed that events occur in repeatable episodes so that the learning algorithm can converge to the optimal solution with sufficient exploration over a large number of iterations. Details of reinforcement learning algorithms are presented in later chapters.

1.7 Contributions

The contributions of this thesis are as follows:

- a test-bed implementation of information-driven sensor selection for indoor target tracking, with a system software architecture design for WSN monitoring, control and visualization
- a distributed sensor election approach with dynamic sampling interval
- an energy efficient data forwarding scheme for multi-hop routing
- a Markov Decision Process framework for non-myopic decision-making, and application of reinforcement learning approximation algorithms

The rest of this thesis is organized as follows. Chapter 2 provides background information for the concepts covered in this thesis, organized into three categories: (i) state estimation for target tracking and information-based approaches for sensor selection, (ii) data routing in wireless sensor networks, and (iii) a decision-theoretic framework based on Markov Decision Processes and reinforcement learning approximation algorithms. In Chapter 3, we describe the design of an indoor target tracking application using ambient sensors, with an adaptive sensor selection scheme, and its implementation in a test-bed, together with our system architecture design for monitoring, control and visualization. Chapter 4 presents a simulation study of distributed sensor election and data routing in multi-hop wireless sensor networks. An MDP formulation is adopted for non-myopic decision-making to choose next-hop neighbor nodes based on minimising the expected sum of costs to the destination node, and approximate solutions based on reinforcement learning are presented. We conclude in Chapter 5 with a summary of this work and propose avenues for future work.

1.8 Summary

In this chapter, the application domain of target tracking with wireless sensor networks was discussed. A general overview of sensor management approaches was presented, addressing energy-efficient and data-centric approaches in sensing, processing and data communication. Different protocols for multi-hop routing were briefly described, along with an introduction to the decision-theoretic and reinforcement learning approaches for non-myopic decision-making. Lastly, the objectives of this work and the organization of this thesis have been presented.

Chapter 2

Background

This chapter provides background information for this thesis. We first describe an overview of state estimation using the discrete Kalman Filter, which consists of recursive predict-update stages, followed by the Extended Kalman Filter (EKF), which is commonly used for state estimation and data fusion implementations. Information utility metrics, that can be used to characterize predicted sensor contributions in terms of information gain, are also described.

Next we review some related routing protocols in wireless sensor networks. The resource-constrained and application-specific nature of wireless sensor networks necessitates energy-efficient and data-centric approaches. We present some illustrative examples of routing protocols from the existing literature. Lastly, we provide an introduction to decision-theoretic frameworks for sensor management, using Markov Decision Processes for decision-making under uncertainty over a long-term discounted horizon. Various formulations are discussed, along with exact, approximate and learning solution aproaches.

2.1 State Estimation and Sensor Selection

2.1.1 An Overview of the Discrete Kalman Filter

This section describes the discrete Kalman Filter, for which the state is estimated, and measurements taken, at discrete points in time, using notation adapted from [22]. The Kalman Filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is assumed to be governed by the linear stochastic difference equation

$$x_k = Ax_{k-1} + Bu_k + w_{k-1}, \tag{2.1}$$

where x_k represents the state variable at time step k, u_k represents the control input and w_k represents the process noise. The observation process is assumed to be of the form

$$z_k = Hx_k + v_k, \tag{2.2}$$

where z_k is the observation of state x_k, and v_k represents the observation noise. The process and measurement noise distributions, denoted by $p(w)$ and $p(v)$ respectively, are assumed to be zero-mean white Gaussian probability distributions that are independent of one another:

$$p(w) \sim N(0, Q), p(v) \sim N(0, R), \tag{2.3}$$

where Q and R represent the variance of the respective distributions.

The Kalman Filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. Thus, the Kalman Filter equations can be categorized into *time-update* (predict) equations and *measurement-update* (update) equations.

In the *predict* phase, the *time update* equations propagate the current state and error covariance estimates in time, to obtain the *a priori* estimates for the

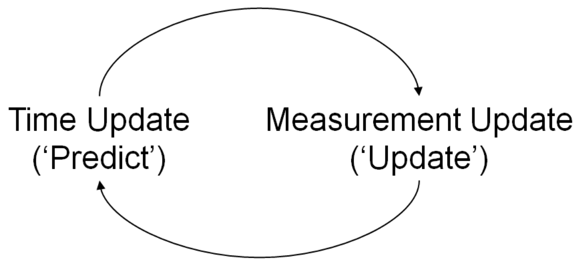


Figure 2.1: The discrete Kalman Filter *predict-update* cycle

next time step. In the *update* phase, the *measurement update* equations provide system feedback by incorporating a new measurement into the *a priori* estimate to obtain an improved *a posteriori* estimate. In this manner, the Kalman Filter recursively *predicts* the state and *updates* it with measurement values, as shown in Figure 2.1.

2.1.2 State Estimation using the Extended Kalman Filter

If the process model and/or the measurement model's relationship with the process model is non-linear, a Kalman Filter that linearizes about the current mean and covariance can be used [22]. This is referred to as an *Extended Kalman Filter* or EKF. The EKF is an approximation that transforms the non-linear relationship to a linearized form using partial derivatives, hence it is a sub-optimal estimate. However, it is suitable and widely used for many real-world applications such as in [23].

In the formulation of the EKF algorithm for tracking applications, the target motion is modeled by the state equation

$$\widehat{X}_{k+1} = F(\Delta t_k)\widehat{X}_k + w_k, \tag{2.4}$$

where X_k is the state of the target at the k-th time step, which consists of the target's location coordinates and/or velocity components, and $\widehat{X}_k$ is the estimate. The duration of the k-th sampling interval is denoted by Δt_k, and the process model is represented by the state propagation matrix $F(\Delta t_k)$ and process noise w_k, which is assumed to be a zero-mean Gaussian probability distribution with variance Q.

Depending on the target application, different propagation models, such as a linear or projectile trajectory within the duration of a sampling interval, or a Gauss-Markov random-walk model [24], can be used to find the posterior estimate $\widehat{X}_{k+1}$ of the target state, given the previous estimate $\widehat{X}_k$. Some applications discretize the infinite state space into regions, such as a grid representation, and develop propagation models in the form of transition probabilities to neighboring regions, or grid squares.

The measurement model is given by

$$z_k = h(X_k) + v_k, \tag{2.5}$$

where h is a (generally non-linear) measurement function dependent on the state X_k, the measurement characteristic (e.g. range, bearing or proximity), and the parameters (e.g. location) of the sensor. v_k denotes the observation noise, which is assumed to have a zero-mean Gaussian distribution with variance R.

The EKF operates in the following way: given the estimate $\widehat{X}_{k|k}$ of the target state $\widehat{X}_k$ at time t_k, with covariance $P_{k|k}$, the predicted state is obtained using the propagation equation

$$\widehat{X}_{k+1|k} = F(\Delta t_k)\widehat{X}_{k|k} \tag{2.6}$$

with predicted state covariance

$$P_{k+1|k} = F(\Delta t_k)P_{k|k}F^T(\Delta t_k) + Q(\Delta t_k) \tag{2.7}$$

The predicted measurement of sensor i is

$$\widehat{z}_{k+1|k} = h(\widehat{X}_{k+1|k}) \tag{2.8}$$

The innovation, i.e. the difference between the actual measurement z_{k+1} of sensor i, and the predicted measurement $\widehat{z}_{k+1|k}$ at t_{k+1}, is given by

$$\Gamma_{k+1} = z_{k+1} - \widehat{z}_{k+1|k} \tag{2.9}$$

with innovation covariance

$$S_{k+1} = H_{k+1} P_{k+1|k} H_{k+1}^T + R_{k+1}, \tag{2.10}$$

where H_{k+1} is the Jacobian matrix of the measurement function h at t_{k+1} with respect to the predicted state $\widehat{X}_{k+1|k}$. The Kalman gain is given by

$$K_{k+1} = P_{k+1|k} H_{k+1}^T S_{k+1}^{-1} \tag{2.11}$$

The state estimate is then updated as

$$\widehat{X}_{k+1|k+1} = \widehat{X}_{k+1|k} + K_{k+1} \Gamma_{k+1} \tag{2.12}$$

and the state covariance is updated as

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1} S_{k+1} K_{k+1}^T \tag{2.13}$$

Figure 2.2 shows an updated illustration of the predict-update cycle from Figure 2.1, with the EKF equations. In addition, there exists a large body of research literature on generalising to non-linear non-Gaussian state estimation for target tracking, and a popular framework is that of *particle filtering* [25], which uses *Monte-Carlo* sampling. A recent comprehensive survey on estimation and infomation fusion techniques can be found in [26].

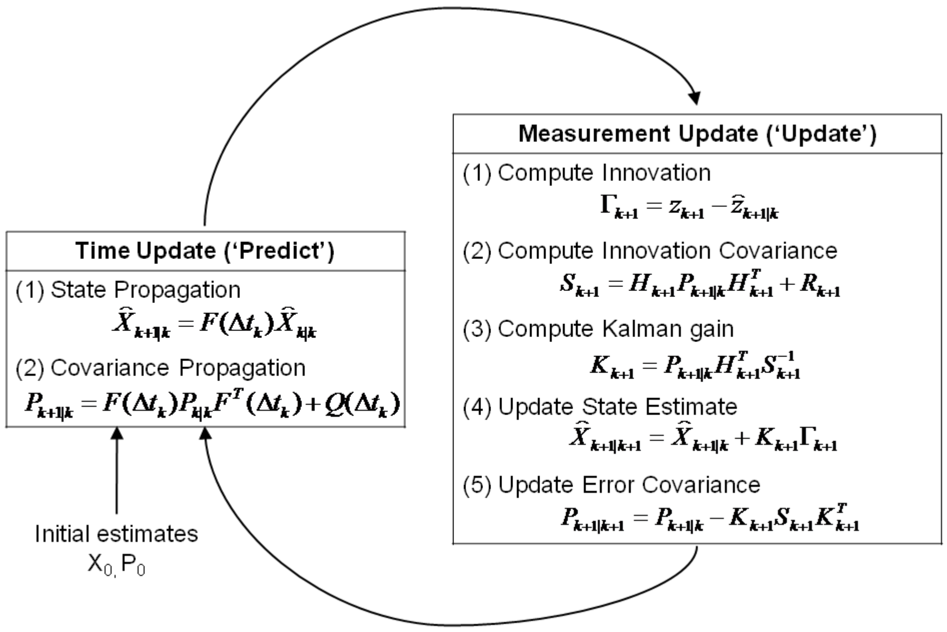

Time Update ('Predict')
(1) State Propagation
$\widehat{X}_{k+1|k} = F(\Delta t_k)\widehat{X}_{k|k}$
(2) Covariance Propagation
$P_{k+1|k} = F(\Delta t_k)P_{k|k}F^T(\Delta t_k) + Q(\Delta t_k)$
Measurement Update ('Update')
(1) Compute Innovation
$\Gamma_{k+1} = z_{k+1} - \widehat{z}_{k+1|k}$
(2) Compute Innovation Covariance
$S_{k+1} = H_{k+1}P_{k+1|k}H^T_{k+1} + R_{k+1}$
(3) Compute Kalman gain
$K_{k+1} = P_{k+1|k}H^T_{k+1}S^{-1}_{k+1}$
(4) Update State Estimate
$\widehat{X}_{k+1|k+1} = \widehat{X}_{k+1|k} + K_{k+1}\Gamma_{k+1}$
(5) Update Error Covariance
$P_{k+1|k+1} = P_{k+1|k} - K_{k+1}S_{k+1}K^T_{k+1}$
Initial estimates
X_0, P_0

Figure 2.2: Operation of the Extended Kalman Filter

2.1.3 Information-driven Sensor Selection

Since the system keeps an estimate of the target state $\widehat{X}_{k|k}$ and associated uncertainty $P_{k|k}$, an *information-utility* measure can be used to quantify the uncertainty of the state estimate as an *information-quality* (IQ) utility metric for sensor selection.

Figure 2.3, adapted from [5], shows the difference between selecting sensors $S1$ and $S2$, where the target state is represented as a Gaussian uncertainty ellipsoid. The objective here is to select the next sensor to result in the largest *reduction* of the estimation uncertainty, and hence provide the largest *information gain*. In Figure 2.3, sensor $S1$ lies along the major axis of the uncertainty ellipsoid, so its observation is able to provide larger uncertainty reduction, and hence more information gain, than sensor $S2$, as evident in its smaller resultant uncertainty ellipsoid. [5] also provides a collection of information-utility

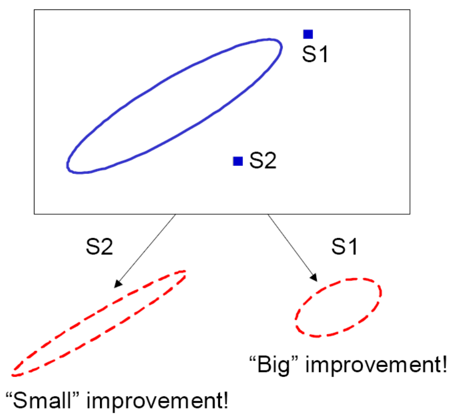


Figure 2.3: Sensor selection based on information gain

measures for target tracking applications, which we briefly review here.

The *Mahalanobis distance* is defined as

$$(x_i - \widehat{x})^T \widehat{\Sigma}^{-1} (x_i - \widehat{x}) \tag{2.14}$$

where x_i is the position of sensor i, $\widehat{x}$ is the mean of the target position estimate, and $\widehat{\Sigma}$ is the error covariance matrix. The Euclidean distance between x_i and $\widehat{x}$ is taken and normalized with $\widehat{\Sigma}$, thus incorporating the state estimate information into the distance measure. The utility function for sensor i, thus, is

$$\varphi(x_i, \widehat{x}, \widehat{\Sigma}) = -(x_i - \widehat{x})^T \widehat{\Sigma}^{-1} (x_i - \widehat{x}) \tag{2.15}$$

An information-theoretic approach can be used to define the *IQ*-measure. The statistical *entropy* measures the randomness of a random variable, and for a discrete random variable x with probability distribution p, it is given by

$$H_p(x) = \sum_{x \in S} p(x) log p(x), \tag{2.16}$$

where S defines the support of the random variable. The smaller the entropy value, the less uncertain the value of the random variable. Hence the information-theoretic utility measure is given by

$$\varphi(x_i, p(x)) = -H_{i,p}(x) \tag{2.17}$$

In fact, the error covariance matrix itself can serve as an *IQ*-measure, since it depicts the size of the uncertainty ellipsoid. Two measures of the *norm* of the covariance matrix are suitable here: the *trace* of the matrix is proportional to the *circumference* of the uncertainty ellipsoid, while the *determinant* of the matrix is proportional to the *volume*.

In addition, the EKF can predict each sensor's potential information-gain before selecting the best sensor and using its observation to make an update. For each sensor i with measurement model

$$z_{i,k} = h_i(X_k) + v_{i,k}, \tag{2.18}$$

its predicted measurement is given by

$$\widehat{z}_{i,k+1|k} = h_i(\widehat{X}_{k+1|k}) \tag{2.19}$$

Sensor i's innovation is not known as its observation is not yet taken. However, its innovation covariance can be predicted by

$$S_{i,k+1} = H_{i,k+1} P_{k+1|k} H_{i,k+1}^T + R_{i,k+1}, \tag{2.20}$$

where $H_{i,k+1}$ is the Jacobian matrix of the measurement function h_i at t_{k+1} with respect to the predicted *a priori* state $\widehat{X}_{k+1|k}$.

The Kalman gain is given by

$$K_{i,k+1} = P_{k+1|k} H_{i,k+1}^T S_{i,k+1}^{-1}, \tag{2.21}$$

and the predicted *a posteriori* state covariance is given as

$$\widehat{P}_{i,k+1|k+1} = P_{k+1|k} - K_{i,k+1} S_{i,k+1} K_{i,k+1}^T \tag{2.22}$$

Thus, the sensor selection objective is to minimize the trace of the predicted *a posteriori* state estimate, $trace(\widehat{P}_{k+1|k+1})$, with the utility function

$$\varphi(x_i, \widehat{X}_{k+1|k}, P_{k+1|k}) = -trace(\widehat{P}_{i,k+1|k+1}) \tag{2.23}$$

In addition to the above-mentioned *IQ*-metrics, other approaches include using *divergence*-measures, such as the *Kullback-Liebler divergence*, to characterize the quality of the state estimate [8], and the *Fisher information matrix* to represent the quality of information available [19]. A review of multi-sensor management in relation to multi-sensor information fusion was presented in [27].

2.2 Routing Protocols in WSNs

Routing protocols for WSNs have been extensively researched, and we choose a few illustrative examples which are more related to this work. Comprehensive surveys of WSN routing protocols can be found in [28], [18].

2.2.1 Data-centric Approaches

In [20], a naming scheme for the data was proposed using attribute-value pairs, which was used by sensor nodes to query for the data on-demand. To create a query, an interest was generated with meta-data, and flooded throughout the network. Nodes were also able to cache the interests and perform in-network data aggregation, which was modeled as a minimum Steiner tree problem. Interest gradients were set up in the reverse direction, based on data rate, duration and expiration time. Using interests and gradients, paths were established between data sources and arbitrary sinks. However, the naming convention was highly application-specific and the periodic propagation of interests and local caching resulted in significant overhead.

In [21], a family of routing protocols was introduced, based on the concept of negotiation for information exchange. Each node upon receiving new data, advertises it to its neighbors and interested neighbors, for which message meta-data is used to reduce redundancies. Neighbor nodes which want the data would reply to the advertisement, to which the current node responds with a DATA reply message. One of the benefits of this aproach is that topological changes are localized since each node needs to know only its single-hop neighbors. However, intermediate nodes, between the data source and an interested querying node, may not be interested in the data, so the querying node may never receive the data it wants. Although data delivery is not guaranteed in the basic scheme, subsequent modifications have addressed this problem [29].

2.2.2 Maximum Lifetime Routing Approaches

In order to address the energy constraints in WSNs, some approaches serve to balance the routing load on the entire network, so as to maximize the network lifetime, which could be defined as the time when the network first becomes partitioned. In [30], the maximum network lifetime problem was formulated as a linear programming problem. This was treated as a network flow problem, and a cost-based shortest-path routing algorithm was proposed, which used link costs that reflected both the communication energy and remaining energy levels at the two end nodes. Simulation results showed better performance than the Minimum Transmitted Energy (MTE) algorithm, due to the residual energy metric.

The approach in [31] consisted of two phases, in which an initial phase of computing and propagating link costs was executed to find the optimal cost paths of all nodes to the sink node, using a back-off mechanism to reduce message exchange. The back-off algorithm sets the total deferral time to be proportional to the optimal cost at a node. Subsequently, the actual data message carried dynamic cost information and flowed along the minimum cost path.

In [32], the authors identified three different routing approaches: (i)minimum-energy routing, which depleted nodes along a good path, (ii)max-min battery level routing, which increased total transmission energy due to detours, and (iii)minimum link cost routing from [30]. These three approaches were formulated as actions within a reinforcement learning framework, in which the states were the sum of energy costs of the minimum-energy path, and the max-min battery life along the path obtained from (ii). The decision-making agent used an on-policy Monte Carlo approach to learn the trade-off parameters between these three candidate schemes, in order to balance the total transmission energy and remaining battery life among nodes.

2.2.3 Information-driven Approaches

An overview for an information-driven approach to sensor collaboration was provided in [5], by considering the information utility of data, for given communication and computation costs. A definition of information utility was introduced, and several approximate measures were developed for computational tractability, along with different representations of the belief state, and illustrated with examples from some tracking applications.

In [33], the authors described the resource constraints in wireless sensor networks, as well as a collaborative signal and information processing (CSIP) approach to dynamically allocate resources, maintain multiple sensing targets, and attend to new events of interest, all based on application requirements and resource constraints. The CSIP tracking problem was formulated within a distributed constrained optimization framework, and information-directed sensor querying (IDSQ) was described as a solution approach. Other examples of combinatorial tracking problems were also introduced.

In [19], the estimation problem for target tracking in wireless sensor networks was addressed using standard estimation theory, by considering the sensor models, associated uncertainties, and different approaches for sensor selection. Information utility measures such as Fisher Information Matrix, covariance ellipsoid and Mahalanobis distance were also described, along with approaches for belief state representation and incremental update. A composite objective function was formulated to trade-off the information utility function with the cost of the bandwidth and latency of communicating information between sensors. Two algorithms were described in detail: Information-directed Sensor Querying (IDSQ) and Constrained Anisotropic Diffusion Routing (CADR), to respectively select which sensors to query, and which to dynamically guide data routing. The implications of different belief state representations were also discussed.

2.3 Decision-theoretic Framework and Algorithms

2.3.1 Markov Decision Processes

Markov Decision Processes (MDPs)[34],[35] are commonly used for decision-making under uncertainty. An MDP consists of a tuple $\langle S, A, P^a_{ss'}, R^a_{ss'} \rangle$, with the following components:

- a set of *states*, S, which represents all the system variables that may change, as well as the information needed to make decisions
- a set of *actions*, A, which represents all the possible actions that can be taken in state $s \in S$
- a state transition probability matrix, in which element $P^a_{ss'}$ represents the transition probability of transiting to state s', from being in state s and taking action a
- a reward matrix, in which element $R^a_{ss'}$ represents the reward of transiting to state s' after being in state s and taking action a

Solution approaches to MDP problems generally try to compute or estimate the value function, which can be represented as functions of state $V(s)$, or state-action pairs $Q(s, a)$. Respectively, they represent the utility of being in state s, or being in state s and taking action a [36], where the utility function is defined based on the optimization objective and the application. The notion of *value* is defined in terms of the expected return, which incorporates the immediate reward, and the expected discounted sum of future rewards under a given policy π. For example, the state value function V and state-action value function Q, under a policy π, can respectively be represented as

$$V^{\pi}(s) = E^{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+1}|s_t = s\right\}, \tag{2.24}$$

and

$$Q^{\pi}(s,a) = E^{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k}r_{t+k+1}|s_t = s, a_t = a\right\}, \tag{2.25}$$

where the discount factor γ reflects diminishing utility of future rewards at the current instance, in order to evaluate the value functions by predicting up to k-steps into the future. Evaluating the expected return as the discounted infinite sum of immediate rewards allows for convergence and mathematical tractability. For situations evaluating either the average-reward or total-reward criterion, Equations (2.24) and (2.25) can be modified by adding an absorbing state with zero reward after the look-ahead horizon of k steps into the future. Details and mathematical proofs are provided in [34].

Some system models for resource management make use of constrained MDPs. For example, in [37], the total network bandwidth is constrained by a theoretical upper bound, and the remaining node energy level has a fixed limit. In [6], the authors try to maximize application performance subject to resource cost, and conversely to minimize resource cost subject to a threshold on application performance metrics.

Target tracking problems have been formulated as partially-observable MDPs, due to the need to estimate the system state from which only partial information from sensors' observations is known. A single target tracking formulation was described in [38], and extended to multi-target tracking in [39]. In [40], multiple available actions were available in each POMDP state as multi radar scans to choose from, for multiple target tracking.

2.3.2 Bellman's Optimality Equations

A fundamental property of the value functions is that they satisfy a recursive relationship. For example, the state value function V^{π} in Equation (2.24) can be written as

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma \; V^{\pi}(s')\right] \tag{2.26}$$

These form the set of Bellman equations for V^{π}, which express a relationship between the value of a state and the values of its successor states. They average over all the possibilities, weighting each by its probability of occurrence [36]. The value function is the *unique* solution to its Bellman equations. In general, solution to MDP problems focus on ways to compute, approximate or learn the value functions of states, V^{π}, or state-action pairs, Q^{π}.

The *Bellman Optimality Equation* is of a similar form:

$$V^{*}(s) = max_{a \in A(s)} \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma \; V^{*}(s')\right] \tag{2.27}$$

The solution to the Bellman Optimality Equation is unique and consists of the solution to the system of equations given by Equation(2.27). Once the optimal value function V^{*} is obtained, any policy that is greedy to V^{*} is an optimal policy:

$$\pi^{*}(s) = \arg max_{a \in A(s)} \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma \; V^{*}(s')\right] \tag{2.28}$$

2.3.3 Dynamic Programming

Dynamic Programming [34],[41] provides a collection of algorithms for solving exactly for the optimal policies, assuming knowledge of a complete model of the environment. They are well developed mathematically and are proven to converge [34]. We briefly review two approaches: value iteration and policy iteration.

Value Iteration

Value iteration consists of recursively updating the value function until no further changes occur, ie. the value functions converge:

$$V_{k+1}(s) = max_{a\in A(s)} \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V_k(s')\right] \tag{2.29}$$

In practice, convergence to within a small neighborhood between successive iterations of the value function $|V_k(s) - V_{k+1}(s)|$ for some small positive value, θ, is a sufficient stopping criterion. The pseudo-code for value iteration, adapted from [36], is shown here:

Algorithm 1: Value Iteration

```
Initialize V arbitrarily, e.g. V(s) = 0 $\forall s \in S$
while $\Delta \geq \theta$ (a small positive number) do
    $\Delta \leftarrow 0$
    for $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow max_a \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V(s')]$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
    end
end
```

Policy Iteration

Policy iteration consists of two simultaneous, interacting processes. *Policy evaluation* attempts to make the value function consistent with the current policy, by iteratively updating the value functions until the stopping criterion is reached, similar to value iteration. *Policy improvement* chooses each action to be greedy with respect to the current value function. As the value functions are iteratively updated, and greedy actions are being simultaneously chosen, the two processes converge to the optimal value function and optimal policy [36]. The pseudo-code for policy iteration is shown next:

Algorithm 2: Policy Iteration

```
1. Initialization Set arbitrary values for all $V(s)$ and $\pi(s)$ $\forall s \in S$
2. Policy Evaluation
while $\Delta >= \Theta$(a small positive number) do
    $\Delta \leftarrow 0$
    for $s \in S$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$
        $\Delta \leftarrow max(\Delta, |v - V(s)|)$
    end
end
3. Policy Improvement
$policy - stable \leftarrow true$
for $s \in S$ do
    $b \leftarrow \pi(s)$
    $\pi(s) \leftarrow \arg max_a \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$
    if $b \neq \pi(s)$ then
        $policy - stable \leftarrow false$
    end
end
if $policy - stable$ then
    stop
else
    go to step 2 - Policy Evaluation
end
```

2.3.4 Monte Carlo Approximation

In the absence of a complete and accurate environment model, dynamic programming methods are of limited applicability. However, MDPs can still be solved approximately by taking sample actions in each state, and averaging over the returns of all episodes that visited that state. This approach is called *Monte-Carlo approximation* – it solves MDPs by approximating the value function with sampling and averaging. Here, it is useful to know the value of taking an action a in state s, so the state-action value function, Q^π, is used instead of the state value function, V^π. The recursive form of the Q-function, Q^π, is

$$Q^\pi(s,a) = \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma \; \pi(s',a')Q^\pi(s',a')] \tag{2.30}$$

The *Bellman Optimality Equation* for Q^* is

$$Q^*(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma \; max_{a'} Q^*(s',a')\right] \tag{2.31}$$

Equation (2.31) forms a set of equations, one for each state-action pair, so if there are S states and A actions, then there are SxA equations in SxA unknowns. Similar to Equation(2.28), once the optimal value function Q^* is obtained, any policy that is greedy to Q^* is an optimal policy:

$$\pi^*(s) = \arg max_a Q^*(s,a) \tag{2.32}$$

In some MDPs, only a small subset of states are ever visited, so Monte-Carlo approximation can discover and utilize sample *trajectories* through the solution space. However, being a sampling method, Monte-Carlo approximation would only be assured to converge to the Bellman Optimality Equations if sufficient *exploring* of the solution space is maintained, in constrast with the greedy policy of *exploiting* the best experienced action for a given state. This is known as the *exploitation-exploration* dilemma. One way to ensure sufficient exploration is to use an ϵ-greedy policy [36], in which a random action is chosen with a small positive probability, ϵ, that decreases with the number of iterations.

Many works in related literature have used Q-value approximation for target tracking applications. In [38], the authors formulated the tracking problem as a *partially-observable* MDP (POMDP), and converted it into a fully observable MDP by defining the problem state in terms of its *belief* state, the conditional probability distribution given the available information about the sensors applied and the measurement data acquired. Particle filtering was used to provide the samples needed for Q-value approximation of candidate actions, and the authors used a cost function that consists of sensor cost and tracking error. This method was extended to track multiple targets in [39]. In [40], the action space was expanded to allow for selecting a combination of multiple sensors. The authors proposed a highsight optimization approach, to address the uncertainties in state transitions as a result of choosing different sensor combinations as actions. The solution was Monte-Carlo approximation with a base policy rollout over a receding finite horizon.

2.3.5 Reinforcement Learning

In general, MDPs face two challenges for their application to real-world problems, (i) the curse of modeling, which is the difficulty of accurately modeling the system and knowing complete information, and (ii) the curse of dimensionality, in that the state and action space grows exponentially with the application's size and complexity. To address this, Reinforcement Learning methods [36],[42],[43] are commonly used in practice for their relative simplicity and their ability to learn from interaction with the environment. Reinforcement Learning approaches differ from Supervised Learning, in that there is no *teacher* to provide the correct *output* for computation of an error signal to provide feedback.

In reinforcement learning, the decision-making agent learns to make decisions by interacting with its environment and learning from experience, to select the best action a given any state s, by obtaining feedback from the environment in the form of a reward signal, $R^a_{ss'}$. The agent learns the Q-function of state-action pairs, which is the sum of expected rewards over some horizon. Specifically, temporal-difference (TD) methods are able to perform incremental updates at the next time-step in the current episode, instead of waiting til the end of that episode, as Monte-Carlo approximation does. This works well for updating the value functions while making online decisions, and also for long episodes, which pose the problem of *credit assignment*, for which it is difficult to identify which actions taken in which states have more weight in contributing to the reward at the end of each learning episode.

Temporal Difference Learning

In *Temporal-Difference* (TD) methods, the next time-step in the current episode is used to provide an update, so that incremental online learning, based on updating Q-values, can be performed. At t_k, for a state-action pair $Q_k(s, a)$, TD-learning makes use of the current model to estimate the next value $Q_k(s', a')$. At t_{k+1}, they immediately form a target and make a useful update using the observed reward r_{k+1} and the estimate $Q_k(s', a')$. The temporal difference between estimated and observed rewards, is fed back into the model to update the Q-value of that state-action pair:

$$Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \left[r_{k+1} + \gamma \; Q_k(s',a') - Q_k(s,a) \right], \tag{2.33}$$

where α is the learning rate, and γ is the discount factor, which indicates how much a future reward is valued at the current iteration k.

Q-learning makes use of past experience with state-action pairs, and a reward or cost signal from the environment in order to learn the Q-function. Hence, potentially-promising state-action pairs that have not been previously explored may be neglected. Hence, in order to guarantee convergence towards optimality, random exploration is introduced in the form of an ϵ-greedy policy[36], where ϵ is a small probability of taking a random action, that is gradually decreased with time, similar to Monte Carlo approximation. Two approaches to Q-learning are briefly described: on-policy and off-policy Q-learning.

On-policy Q-learning

In on-policy Q-learning, actions are chosen based on an ϵ-greedy policy, that is, the best action in the current state is chosen with a probability (1-ϵ), and a random action with probability ϵ. This applies to both the current and predicted state-action pairs, $Q(s, a)$ and $Q(s', a')$ respectively. The update step involves the elements from Equation (2.33) in the form of the tuple $\langle s_k, a_k, r_k, s_{k+1}, a_{k+1} \rangle$. The following pseudo-code for on-policy Q-learning is taken from [36]:

Algorithm 3: On-policy Q-learning

```
Initialize $Q(s, a)$ arbitrarily
for episode $i \leftarrow 1 : maxepisodes$ do
    Initialize $s$
    Choose $a$ from $s$ using $\epsilon$-greedy policy
    for each step $k \leftarrow 1 : maxsteps$ do
        Take action $a$, observe $r$ and $s'$
        Choose $a'$ from $s'$ using $\epsilon$-greedy policy
        Update $Q_{k+1}(s, a)$ with Equation (2.33)
        $s \leftarrow s', a \leftarrow a'$
    end
    until $s$ is terminal
end
```

Off-policy Q-learning

In off-policy Q-learning, the learned action-value function Q directly approximates Q^*, the optimal action-value function, independent of the policy being followed. The current action is chosen according to an ϵ-greedy policy, but the update step makes use of the *best* subsequent action from the Q-function at the current episode. According to [36], this helped to simplify the theoretical analysis of the algorithm and enable early convergence proofs. Q-learning is especially useful in being able to learn an optimal policy, with reference to following an ϵ-greedy policy. The update equation is given by

$$Q_{k+1}(s, a) \leftarrow \; Q_k(s, a) + \alpha \left[r_{k+1} + \gamma \; max_{a'} Q_k(s', a') - Q_k(s, a) \right] \tag{2.34}$$

with the following pseudo-code [36]:

Algorithm 4: Off-policy Q-Learning

```
Initialize $Q(s, a)$ arbitrarily
for episode $i \leftarrow 1 : maxepisodes$ do
    Initialize $s$
    for each step $k \leftarrow 1 : maxsteps$ do
        Choose $a$ from $s$ using $\epsilon$-greedy policy
        Take action $a$, observe $r$ and $s'$
        Update $Q_{k+1}(s, a)$ with Equation (2.34)
        $s \leftarrow s'$
    end
    until $s$ is terminal
end
```

Related Work

In related work, the authors in [7] applied Q-learning to trade-off the costs incured due to sensor deployment or activation, with the rewards from information gain due to collected measurements. The immediate reward was computed using information gain measured by *Renyi*-divergence between predicted and updated probability densities of the state estimate. In [13], the authors used reinforcement learning for distributed task allocation for an object tracking scenario, in which nodes learn to perform sub-tasks such as sampling, communication, aggregation, and powering down to sleep-mode, based on utilities defined by application-specific parameters, such as throughput and energy usage. In [44], reinforcement learning was used to perform sensor scan selection for multiple object tracking, identification and classification of their threat levels, while addressing sensor costs.

In [45], the author proposed many approaches to speed up on-line reinforcement learning, by implementing a CMAC controller with a Hierarchical Mixture of Experts architecture. The author also addressed how to find an exploration strategy and preserve specialised knowledge, and how to do context-dependent learning. CMAC was extended in [24] for energy-efficient target tracking in sensor networks, where the tracking area was divided into clusters. The resource management problem was divided into two portions – to predict the target trajectory and set sampling rates.

At the upper tier, the higher level agent (HLA) had to keep track of the listening cluster and its dwell time, and set the sampling rate by activating the node's status: whether to sense at a long sampling interval, perform tracking with a short sensor sampling interval, or remain idle. It incurred a cost that was a weighted sum of proportional power consumption and proportion of wrong predictions. At the lower tier, the lower level agent (LLA) had to keep track of the cluster and predict the target trajectory, incurring a cost of 0 for correct prediction, and 1 otherwise. The hierarchical MDP was solved by Q(λ)-learning, using CMAC as a neural-network like implementation to approximate the Q-function, which was stored in a look-up table on a WSN mote's Flash memory.

In [6], the authors performed sensor management by choosing sensor subsets and the data fusion centre, which may communicate with the sink along multiple hops. They formulated the resource management problem as a constrained MDP, relaxed the constraints using Lagrangian variables, and solved it by subgradient update and rollout methods. This was based on an earlier approach proposed by Castañon [46], in which a dynamic hypothesis testing and target classification problem was formulated as a Markov Decision Process and solved using approximate dynamic programming with Lagrangian relaxation and policy rollout. This work was extended to multi-hop WSNs in [47].

2.4 Summary

In this chapter, the theoretical background used in this thesis was described. A general description of the Extended Kalman Filter was presented, with information-driven sensor selection using information-utility measures. This provides the background for Chapter 3, which describes the design of an indoor target tracking application and its implementation in a real-world test-bed.

A brief overview of multi-hop routing protocols for wireless sensor networks was also described, followed by an overview of Markov Decision Processes, with an introduction to methods that compute, approximate or learn the value functions to determine an optimal policy. In Chapter 4, we describe the use of reinforcement learning to find a sensor election policy for multi-hop routing.

Chapter 3

Design and Implementation of an Indoor Tracking test-bed

3.1 Introduction

This chapter describes the design and implementation of a wireless sensor network for ambient sensing in an indoor smart space. There are two main components:

1. Implementation of indoor human tracking
2. Integration of smartphone mobile devices for monitoring, control and visualization of WSNs

In the first application, we apply the Extended Kalman Filter, described in the previous chapter, for state estimation and data fusion with ambient sensor observations, with an information-driven sensor selection approach, based on minimizing the trace of the predicted state covariance matrix. The aim of this work is to develop a proof-of-concept test-bed for implementing our resource management algorithms for real-world experimentation and data collection. We describe the design and implementation of the estimation and sensor selection algorithms, and a two-tier architecture for resource management in WSNs, and we provide some comparisons between different sensor selection approaches.

In the second application, we extend the existing test-bed implementation by integrating smartphone mobile devices with our WSN implementation, to create a mobile device layer in our system architecture. Smartphones have grown quickly in popularity and capabilities, allowing access to these ubiquituous devices to perform real-time sensing, processing, communication and data visualization. Using open-source Google Android OS, we develop an application for real-time sensor network monitoring, control and visualization, and we deploy it in our WSN test-bed implementation. Integrating smartphones with WSNs holds significant potential for future new applications, such as indoor target tracking, activity monitoring, pervasive computing and real-time participatory sensing [48],[49].

3.2 Background

3.2.1 Hardware Platforms

Wireless Sensor Networks consist of low-power devices with limited sensing, processing and radio communication capabilities. Many research prototypes currently exist, and *commercial-off-the-shelf* (COTS) systems are available from companies such as Crossbow Technologies [50], and EasySen [51].

Popular development platforms from Crossbow, such as the TelosB and MICAz platforms, run on low-power microcontrollers at processing speeds of up to 8MHz. Intel's iMote2 platform features a much faster processor running up to 400MHz, with dynamic voltage scaling capabilities for power conservation. For radio communications, current WSN platforms include a radio transceiver (Texas Instruments CC2420), that implements a simplified version of the IEEE 802.15.4 standard for low-power Personal Area Networks (PANs). The wireless motes communicate in the 2.4GHz frequency band over 26 possible radio channels at a maximum data rate of 250 *kbps*, and they make use of the Carrier Sense Multiple Access (CSMA) protocol for wireless medium access control. Figure 3.1 shows images of the radio boards for the TelosB, MICAz and iMote2 platforms.

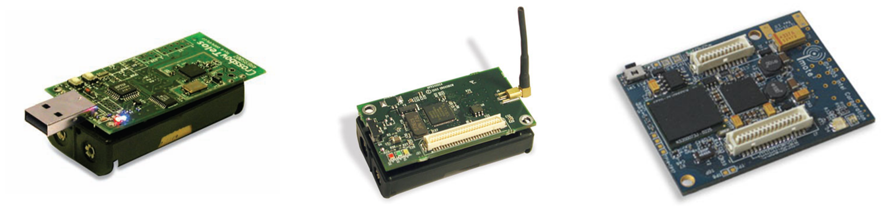

(a) TelosB (b) MICAz (c) iMote2

Figure 3.1: COTS WSN Mote Platforms

These WSN development platforms come with basic sensor boards with temperature, humidity, light and accelerometer sensors, and provide communication interfaces to connect more sensors or communicate with other devices. For example, the MICAz prototyping board exposes communication interfaces such as the Serial Peripheral Interface Bus (*SPI*) for synchronous communications, and an Inter-Integrated Circuit interface (I^2C) for communication with external sensors, such as ultrasonic sensors. The iMote2 platform supports advanced connection interfaces, such as a CIF interface for an image camera.

Figure 3.2: COTS Stargate WSN Gateway

In addition, more advanced single-board computer platforms are widely used to provide more processing power in WSNs and they can function as gateways to 802.11 Wi-Fi or wired backbone networks. Figure 3.2 shows a Crossbow

Stargate Gateway which runs embedded LINUX OS on a 400MHz processor, and provides a variety of communication interfaces such as an Ethernet port, a Wi-Fi Compact Flash slot, USB ports, and a MICAz connector for interfacing with MICAz motes, as shown in Figure 3.3. Hence, Stargate-class nodes can serve as a communication gateway between WSNs over 802.15.4 radio, Wi-Fi ad-hoc networks over 802.11 radio, and wired networks over Ethernet LAN.

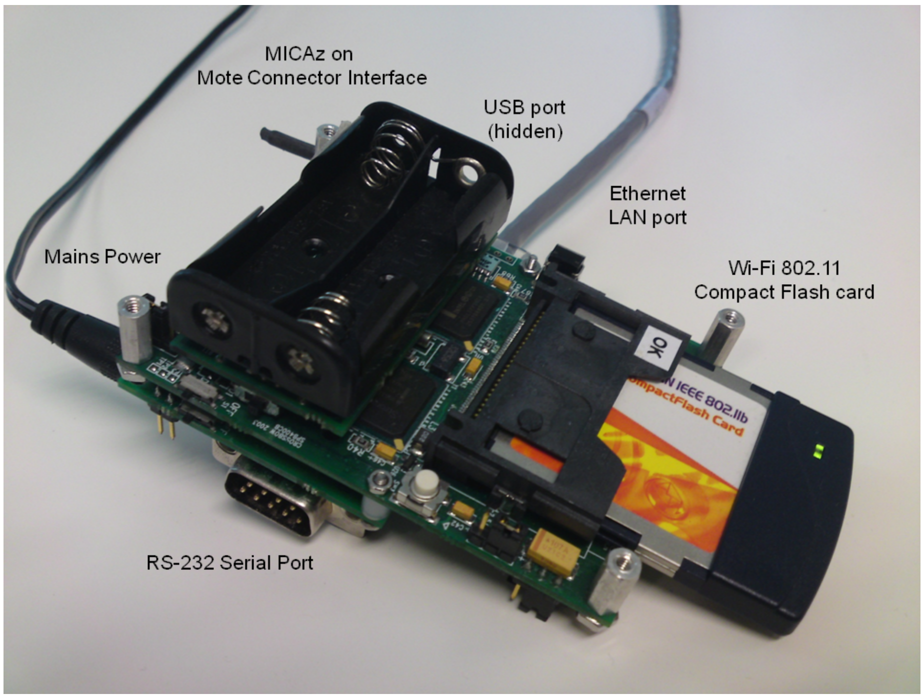


Figure 3.3: Stargate WSN Gateway with communication interfaces

Using LINUX OS, Stargate gateways are able to support multiple processes in concurrent threads, thus allowing them to simultaneously execute multiple tasks such as data-intensive processing, WSN control and resource management, and communication gateway functions across multiple interfaces. Using socket communications, Stargate gateways allow client applications to remotely access streams of observation data, or to execute queries in WSN nodes. Due to higher power requirements, Stargate gateways are powered by mains power supply, and commonly used to manage battery-operated WSN motes. With their superior processing capabilities and communication interfaces, Stargate nodes usually serve as cluster heads to coordinate and manage clusters of low-power WSN sensor mote devices.

3.2.2 WSN Software

TinyOS [52] is an open-source operating system for low-power WSNs and it is widely used as the *de facto* OS for developing WSN applications, with a large and growing repository of research projects. TinyOS features a component-based software architecture made up of software *modules* which are linked together using *configuration* files via *interface* definitions, all specified in the *nesC* language with *C*-like syntax. TinyOS supports the IEEE 802.15.4 standard, and its component library includes network protocols, low-level hardware access (such as timers and schedulers), sensor drivers, and data acquisition routines. At compile-time, only the required components are linked from the component library, so as to reduce the memory footprint and code size.

TinyOS features an event-driven execution model to improve on power management, and yet allow flexibility in scheduling tasks interacting with unpredictable physical and wireless communication events. Its *split-phase* operation makes use of a *command–event* interface which components use to trigger one another, so that minimal processor time is spent waiting. When an event triggers a low-power interrupt, the node wakes up, schedules an appropriate task, and returns to low-power sleep mode, thus conserving energy. This approach minimizes time spent waiting for lost events or wireless packets, which is assumed to be a common occurance in WSNs. Higher-level applications are designed for data redundancy and fault tolerance in order to ensure desired application-level performance, such as delay and detection accuracy.

3.3 System Overview

3.3.1 System Flowchart

Figure 3.4 shows a flowchart illustrating how the EKF and sensor selection algorithms, described in Chapter 2, are implemented in our indoor target tracking test-bed. Once the system is started, it stays in an idle state, in which nodes periodically sense, for example every 100ms, whether there is any target in the room. When a target is detected by a sensor node, the node initializes the state estimate X_0 to its location coordinates, and the covariance matrix P_0 to some large values which reflect the large initial uncertainty of the state estimate.

Based on the state estimate, the current node computes the prior state $\widehat{X}_{k+1|k}, \widehat{P}_{k+1|k}$ using the prediction equations (2.6) and (2.7) with the process model $F(\Delta t_k)$, where Δt_k is the update interval at step k. The prior state estimate is then used by the current node to select the next sensor node based on the IQ metric, $IQ_i = trace(\widehat{P}_{i,k+1|k+1})$, for each node i, using the equations (2.19) to (2.23). For the case when the target is first detected, there is no velocity information to predict the prior state estimate, so the current sensor node randomly selects the next sensor node from a set of candidate sensors, for which the target lies within the detection range of each of them.

Subsequently, the current node passes the prior state estimate information $\widehat{X}_{k+1|k}, \widehat{P}_{k+1|k}$ to the selected node which takes a measurement z_{k+1}. If the selected node detects the target, it updates with its measurement to obtain the posterior state estimate $\widehat{X}_{k+1|k+1}, \widehat{P}_{k+1|k+1}$, using the equations (2.9) to (2.13), and makes a prediction to compute the next prior estimate. If the selected node is unable to detect the target, it checks whether the number of target misses has exceeded a threshold value. If so, the target is assumed to be lost and the system returns to the idle state. Otherwise, the selected node proceeds to predict the next prior state estimate for step $k+1$. The EKF and sensor selection algorithm continue to track the target in recursive *predict* and *update* stages.

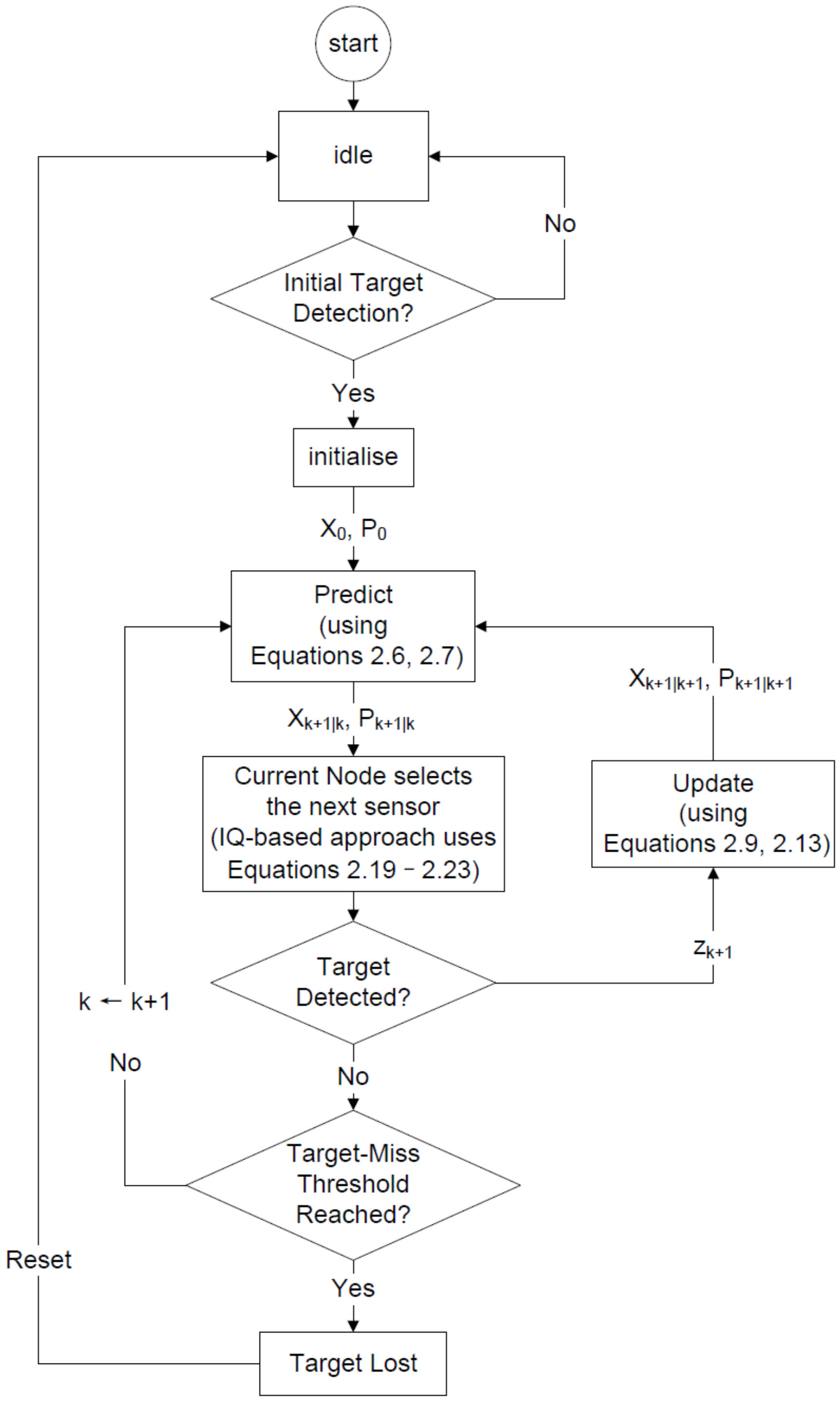

start
idle
No
Initial Target Detection?
Yes
initialise
X_0, P_0
Predict (using Equations 2.6, 2.7)
$X_{k+1|k}$, $P_{k+1|k}$
Current Node selects the next sensor (IQ-based approach uses Equations 2.19 - 2.23)
$X_{k+1|k+1}$, $P_{k+1|k+1}$
Update (using Equations 2.9, 2.13)
z_{k+1}
Target Detected?
k ← k+1
No
No
Target-Miss Threshold Reached?
Reset
Yes
Target Lost

Figure 3.4: Flowchart for State Estimation and Sensor Selection

3.3.2 System Models

This section describes the system models that we use in our simulations and test-bed implementation. The process model is represented by a transition matrix $F(\Delta t_k)$, where Δt_k is the update interval at step k. We consider a target moving in a 2-d plane, so the target state is defined as $X_k = (x_k, v_{x,k}, y_k, v_{y,k})$ at time step k, where x_k and y_k are the x- and y-coordinates of the target, and $v_{x,k}$ and $v_{y,k}$ are the velocities of the target along the x- and y-directions. Recall from equations (2.4) and (2.5), that the process and observations models are, respectively

$$\begin{aligned} \widehat{X}_{k+1} &= F(\Delta t_k)\widehat{X}_k + w_k && (3.1) \\ z_k &= h(X_k) + v_k && (3.2) \end{aligned}$$

The Constant Velocity and Integrated Ornstein-Uhlenbeck process models are described next.

Constant Velocity Process Model

A constant-velocity process model assumes that the target's trajectory within an update interval is linear, i.e. it travels at a constant velocity. The matrix representation is:

$$F(\Delta t_k) = \begin{bmatrix} 1 & \Delta t_k & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t_k \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

and the process noise covariance matrix is

$$Q(\Delta t_k) = q \begin{bmatrix} (\Delta t_k)^3/3 & (\Delta t_k)^2/2 & 0 & 0 \\ (\Delta t_k)^2/2 & \Delta t_k & 0 & 0 \\ 0 & 0 & (\Delta t_k)^3/3 & (\Delta t_k)^2/2 \\ 0 & 0 & (\Delta t_k)^2/2 & \Delta t_k \end{bmatrix} \tag{3.4}$$

Integrated Ornstein-Uhlenbeck Process Model

The Integrated Ornstein-Uhlenbeck (IOU) process model [53] is a nearly constant-velocity model that bounds the Brownian velocities, preventing them from growing excessively large, when there are missed detections [8]. In practical deployments, detection misses may occur due to missed observations, such as when ultrasonic pulses are reflected off an uneven target surface and are missed at the receiver transducer, or due to wireless packet loss. As the velocity component of the target's state estimate is now more uncertain, and has less contribution to the state propagation, the process noise is increased. The IOU process model matrices are:

$$F(\Delta t_k) = \begin{bmatrix} 1 & \Delta t_k & 0 & 0 \\ 0 & F_v & 0 & 0 \\ 0 & 0 & 1 & \Delta t_k \\ 0 & 0 & 0 & F_v \end{bmatrix}, \tag{3.5}$$

where

$$F_v = e^{-\eta \Delta t_k}, \tag{3.6}$$

and

$$Q(\Delta t_k) = q \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \Delta t_k (1 - F_v)^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Delta t_k (1 - F_v)^2 \end{bmatrix} \tag{3.7}$$

Observation Model

The observation model is that of a range sensor, corrupted by noise, that is assumed to follow a zero-mean Gaussian probability distribution.

$$z_i(\widehat{X}_k) = \|(x_i, y_i) - (x_k, y_k)\| + v_{i,k}, \tag{3.8}$$

where x_i and y_i respectively are the x- and y- coordinates of sensor i. In our application, we make use of ultrasonic sensors which can interfere with one another, with an interference region that is significantly larger than the detection region. Hence only one sensor is allowed to fire at any one time. As depicted in Figure 3.4, the predicted state is used to find the sensor with the least predicted value of $trace(\widehat{P}_{i,k+1|k+1}))$, which provides the highest utility and is selected as the next sensor to activate. We term this approach as *adaptive sensor selection*.

3.4 Simulation Study

3.4.1 Sensor Deployment

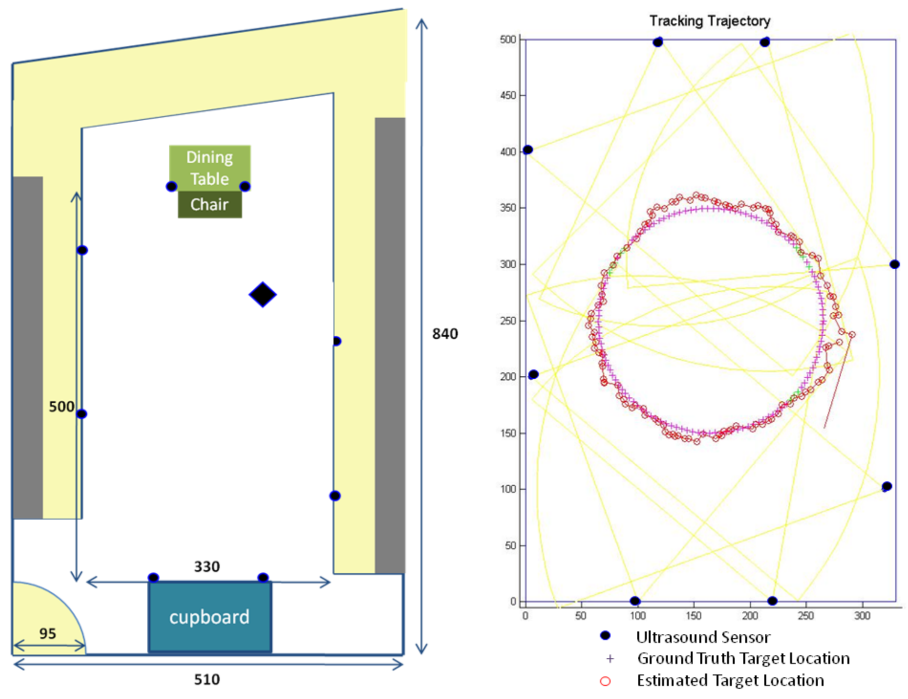
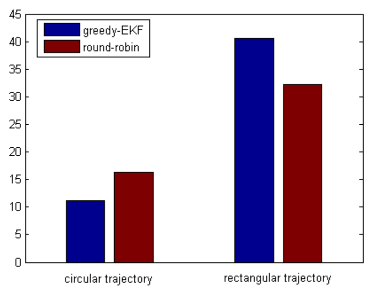


Figure 3.5: Test-bed Sensor Deployment and Sensor Coverage

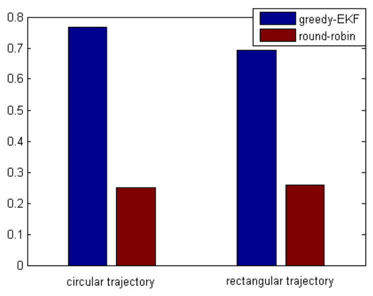
Figure 3.5 shows the sensor deployment in the test-bed. The left diagram shows the sensor deployment with respect to the room dimensions and other furniture, while the right diagram shows the sensor coverage. Sensor orientations were carefully chosen with calibrated sensor measurements to maximise the sensor coverage in the room, based on the ultrasonic sensors' conical detection region. An example simulation is shown in the right diagram for a circular target trajectory, along with the ground truth target location and the estimated location from the EKF.

3.4.2 Simulation Results

Based on our test-bed deployment, sensor measurements were obtained, along with ground truth location traces of a moving human target, and used to perform simulations to study IQ-driven sensor selection. We simulated greedy-EKF, an adaptive sensor selection scheme which selected the sensor that minimized the trace of the predicted EKF error covariance matrix. The greedy-EKF and round-robin sensor selection schemes were compared based on tracking error and detection ratio, which we define as the number of detections over the total number of sampling intervals for each experiment. Circular and rectangular target trajectories were simulated for 500 runs each, using a constant-velocity process model in the EKF algorithm, with a target velocity of $50cms^{-1}$.



(a) Tracking Error

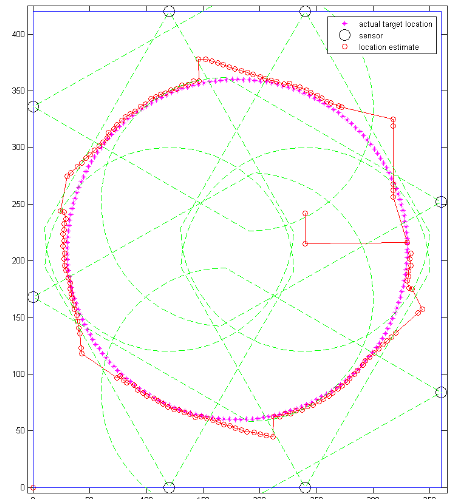


(b) Detection Ratio

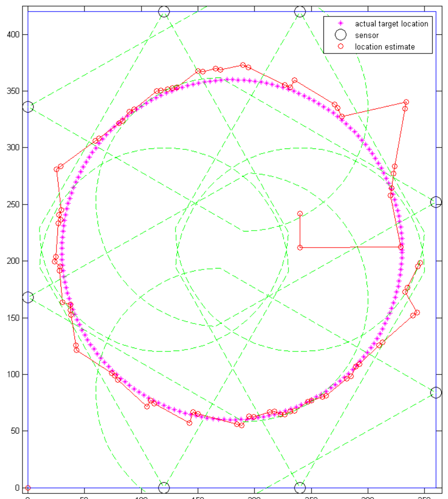
Figure 3.6: Comparison between adaptive sensor selection and round-robin (constant velocity process model)

Figure 3.6 shows that for the circular trajectory, the greedy-EKF approach had lower tracking error and higher detection rate than round-robin sensor selection, as greedy-EKF used the EKF state estimate to more accurately predict the target's location and the sensors' predicted information gain.

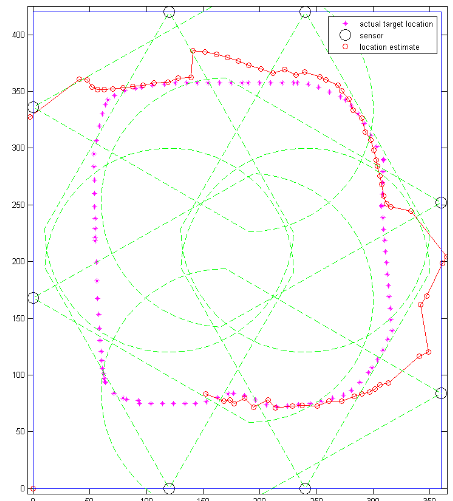
On the other hand, for the rectangular trajectory, the constant-velocity process model was shown to be unable to accurately predict the rectangular trajectory, as seen in the increase in tracking errors for both schemes. In addition, the greedy-EKF approach had a significantly larger increase in tracking error than the small decrease in its detection rate, as compared to round-robin. Figure 3.7



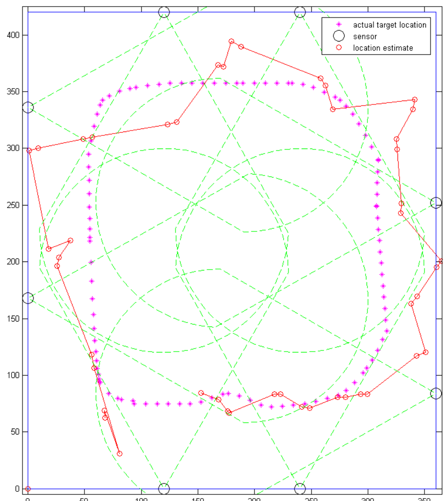
(a) Circular Trajectory - Adaptive



(b) Circular Trajectory - Round-robin



(c) Rectangular Trajectory - Adaptive

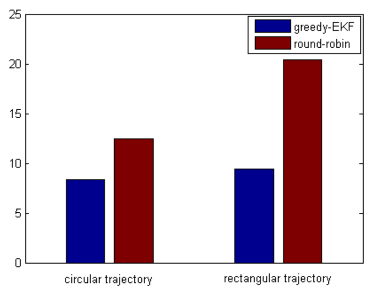


(d) Rectangular Trajectory - Round-robin

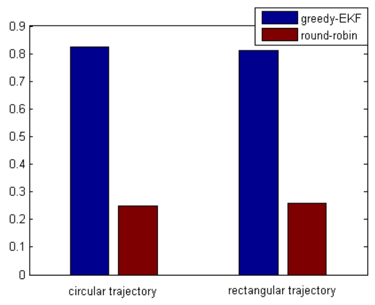
Figure 3.7: Comparison of sensor selection approaches for circular and rectangular trajectories (constant velocity process model)

shows target location estimate plots for both circular and rectangular trajectories. For the circular trajectory, greedy-EKF out-performed round-robin, but for the rectangular trajectory, greedy-EKF was unable to recover the state estimate when the target was lost, while the fixed round-robin selection policy was able to recover the target after some time. The greedy-EKF approach, while attempting to better predict which sensor to use, was very prone to wrong predictions as a result of less detections, as shown by the sparse location updates, and the significantly increased tracking error.

Figure 3.7 also illustrates the impact that the sensor detection rate had on the tracking algorithm, as frequent updates were needed for reasonable tracking accuracy. Under the constant-velocity process model, the target's location estimate was assumed to follow a straight line, so missed detections had a large impact on the tracking accuracy, as well as large discontinuities in the estimated trajectory. In Figure 3.7(d), pairs of consecutive state estimates were observed, followed by large jumps to the next location estimate. The consecutive estimates were due to the round robin sensor selection scheme, where instead of choosing the next sensor to maximise information gain, a fixed sequence of sensors was used, resulting in small information gain and small displacements in the location estimate followed by large discontinuities as the EKF algorithm tried to recover the target state estimate.



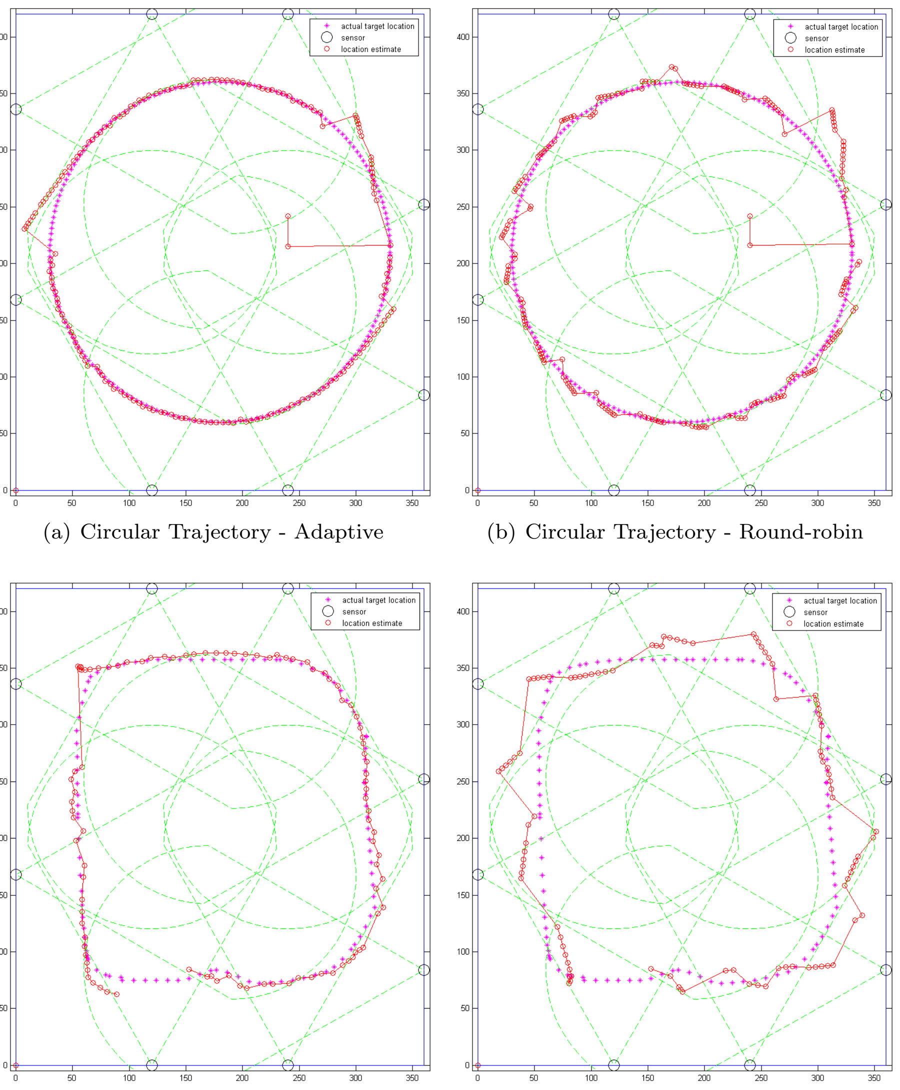
(a) Tracking Error



(b) Detection Ratio

Figure 3.8: Comparison between adaptive sensor selection and round-robin (IOU process model)

Figure 3.8 shows our results from 500 simulations runs, using the IOU process model with decay factor $\eta = 0.9$ for bounding the Brownian velocity. For both trajectories, adaptive sensor selection out-performed the round-robin scheme in terms of less tracking error and higher detection ratio. Furthermore, the IOU process model had less tracking error compared to the constant-velocity process model. From Figure 3.9, using the IOU process model produced smoother tra-



(a) Circular Trajectory - Adaptive

(b) Circular Trajectory - Round-robin

(c) Rectangular Trajectory - Adaptive

(d) Rectangular Trajectory - Round-robin

Figure 3.9: Comparison of sensor selection approaches for circular and rectangular trajectories (IOU process model)

jectory plots with significantly less 'jumps' and discontinuities. This is because the IOU model restricted predicting ahead using the estimated velocity when the target was lost, until the target was detected again. Using this process model allowed the EKF to track the target better in the presence of missed detections, which may be caused by sensor interference or wireless packet loss.

3.5 Test-bed Implementation

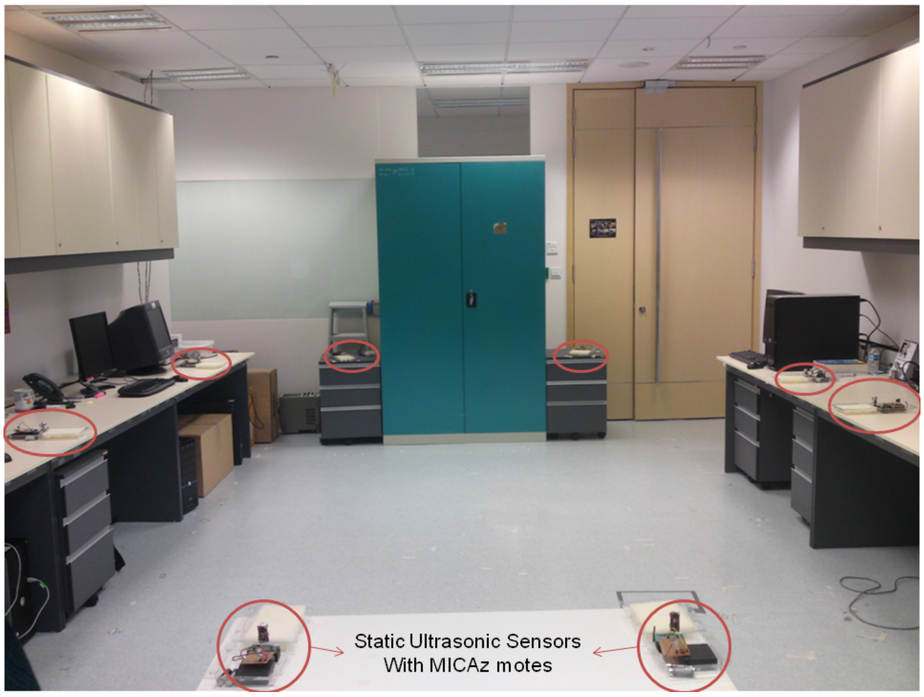


Figure 3.10: Deployed test-bed in an indoor smart space

Figure 3.10 shows a snapshot of our test-bed deployment in an indoor smart space. Static ambient sensors, interfaced to WSN motes, are deployed around the room and highlighted in red ovals. WSN nodes in the smart space are able to overhear one another over IEEE 802.15.4 radio, and the sensor nodes are organized into a single-hop star network topology.

3.5.1 Clustered System Architecture

Figure 3.11 shows the design of our tracking cluster architecture. Our test-bed is regarded as a logical cluster of WSN motes, managed by a WSN Stargate Gateway, which has more processing resources and communication interfaces. Intra-cluster wireless communication takes place over 802.15.4 radio, based on a CSMA MAC protocol in a single-hop topology. Inter-cluster communication takes place over TCP socket connections, over wired Ethernet LAN or wireless Wi-Fi. Application information from multiple smart-spaces (clusters) is sent to a centralized server, for further processing, storage and visualization. Users of the tracking application can set up client socket connections to obtain visualization output streams from the centralized server, or streams of tracking coordinates directly from the Stargate Gateways. The implementation of this clustered architecture would be part of our future work.

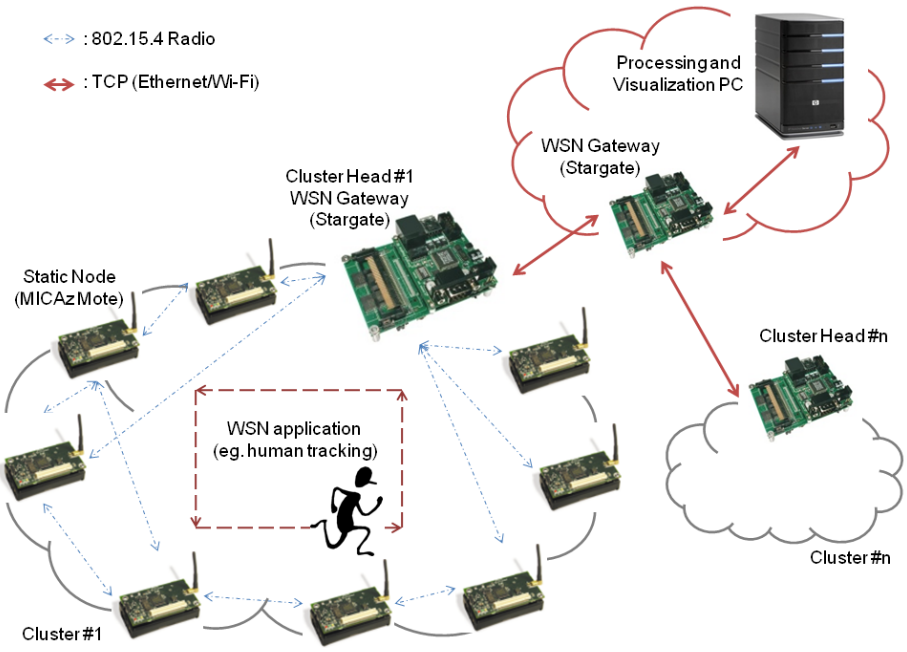


Figure 3.11: Clustered System Architecture

3.5.2 System Visualization

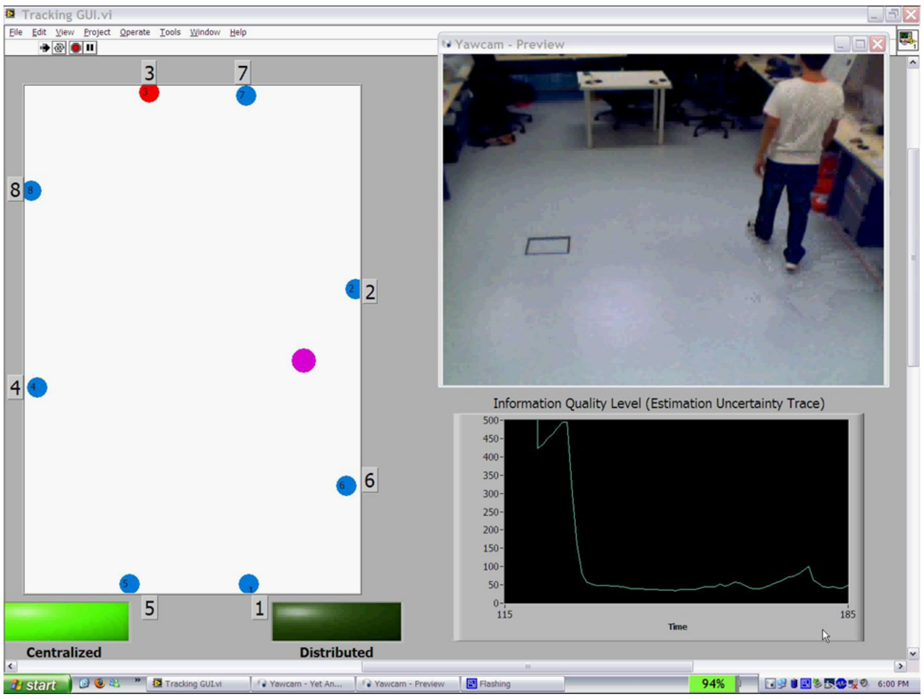


Figure 3.12: Visualization and User Interface

Figure 3.12 shows the visualization and user interface of our indoor tracking application. The left part of the figure illustrates the test-bed with the static sensors represented by blue circles, which turn red when the corresponding sensor is selected. The human target's real-time location estimate is represented as a purple circle moving around in the test-bed, along with a live video feed at the upper right corner of the visualization display, for ground-truth comparison. The lower right display shows the covariance trace, which represents the estimation uncertainty. From an initial high level of uncertainty when the tracking system was initialized, the covariance trace decreases greatly and stays low as the human target is being tracked by the ultrasonic sensors in the test-bed.

3.6 Integrating Mobile Devices with WSNs

In this section, we incorporate smartphone mobile devices into our WSN testbed implementation and add a mobile device layer, consisting of mobile computing devices inter-connected over an ad-hoc wireless network, into our system architecture. Mobile devices can also communicate with Stargate WSN Gateways as peers over Wi-Fi radio. We develop smartphone applications for real-time monitoring, control and visualization of WSNs over interactive user interfaces, incorporate our indoor human tracking WSN application into a smartphone application, and demonstrate it over a software emulator and on an actual Android-based smartphone.

Smartphones have been used in healthcare and activity recognition applications [54],[55] and personal environmental impact sensing and monitoring [48]. Providing smartphones with access to real-time data from ambient sensors holds significant promise in improving the development and deployment of WSN applications. In addition to being a useful tool to obtain network statistics and sensor measurements in real-time to test resource management algorithms and networking protocols, users can interact with smartphone applications, for example, to perform indoor self-localization with the help of ceiling-mounted range sensors. This provides input data streams for users to obtain sensor data to self-monitor their daily activities, which is useful for elderly healthcare and health monitoring applications, so as to complement some of the related works mentioned above, or to provide functionalities for new mobile applications.

3.6.1 Mobile Device Platforms

Smartphone mobile devices have been growing in popularity due to increased functionality from mobile applications and location-based services, integration of multiple on-board sensors and decreasing costs. Most smartphones are equipped with sensors such as an on-board accelerometer to detect acceleration, a digital compass, a built-in microphone and speaker, a camera and a GPS receiver. The

HTC Dream smartphone that we use for our implementation uses a Qualcomm processor with a processing speed of 528 MHz, with 256 MB of ROM and 192 MB of RAM, so it is able to run significant processing tasks. Besides the basic communication capabilities of text messaging, voice and video calls over cellular networks, smartphones also come equipped with Wi-Fi for wireless data access and mobile Internet browsing, and Bluetooth chipsets to communicate with other sensors and devices.

There has been rapidly growing interest in using smartphones to sense and keep track of location and environment information, personalized to the user's activities throughout the day. Location information can be provided by a few sources, such as GPS, Wi-Fi and cellular proximities in telco networks. Users can make use of location information to find nearby landmarks and events-of-interest, to be kept notified of friends in the vicinity, and to track their location and activity patterns. Similarly telco operators aim to provide more useful location-based services to provide advertisements and recommendations, and support streaming media and social networking applications.

3.6.2 Android OS

Android is an open-source software stack, developed by Google for mobile devices, that includes an operating system, middleware and key applications [56]. Android uses a version 2.6 Linux kernel and runs on a Dalvik virtual machine, that is optimized for mobile devices. Its application framework enables reuse and replacement of software components on the mobile phone for resource efficiency, and it provides tools for graphics handling, media support and SQLite for structured data storage.

In addition to basic GSM telephony functions, Android OS has hardware support for Bluetooth, 3G, and Wi-Fi communications, as well as a variety of sensors, such as camera, GPS, compass, and accelerometer. Android provides a common software OS abstraction from the device-specific hardware components, so that mobile device applications are able to work on all phone devices that support the Android OS platform. The Android SDK provides the tools and interfaces for developing applications on the Android platform for mobile devices using the Java programming language, and it also provides a device emulator and tools for debugging, memory and performance profiling.

3.6.3 Extended System Architecture

In this section, we propose an extension of the two-tier cluster architecture proposed in section (3.5.1) to include the mobile device layer. Figure 3.13 shows the proposed architecture integrating mobile devices and WSNs.

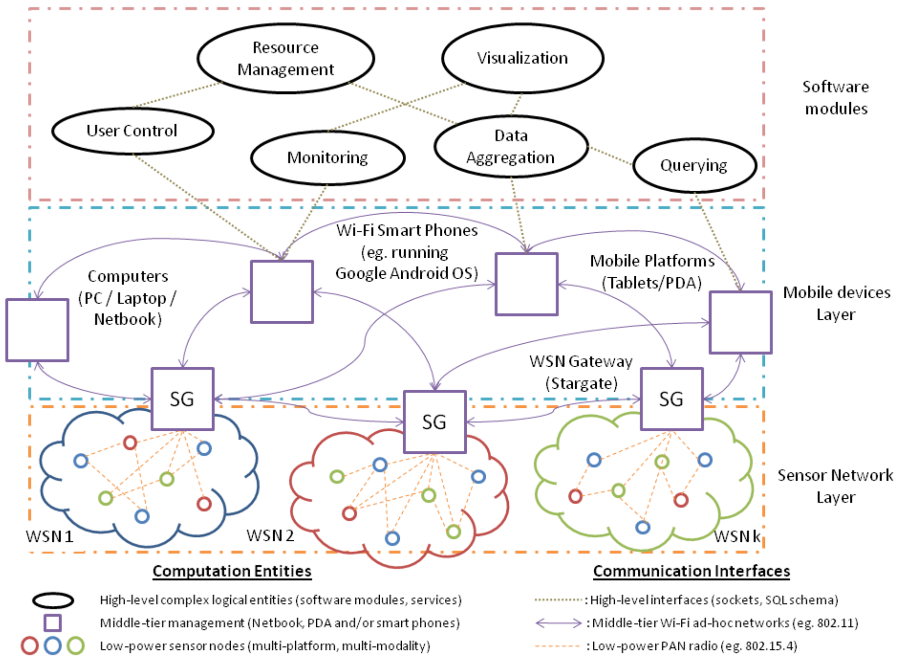

Resource Management
Visualization
Software modules
User Control
Monitoring
Data Aggregation
Querying
Wi-Fi Smart Phones (eg. running Google Android OS)
Computers (PC / Laptop / Netbook)
Mobile Platforms (Tablets/PDA)
Mobile devices Layer
WSN Gateway (Stargate)
SG
SG
SG
Sensor Network Layer
WSN 1
WSN 2
WSN k
Computation Entities
High-level complex logical entities (software modules, services)
Middle-tier management (Netbook, PDA and/or smart phones)
Low-power sensor nodes (multi-platform, multi-modality)
Communication Interfaces
: High-level interfaces (sockets, SQL schema)
: Middle-tier Wi-Fi ad-hoc networks (eg. 802.11)
: Low-power PAN radio (eg. 802.15.4)

Figure 3.13: Software Architecture integrating mobile devices and WSNs

The bottom layer in this architecture is the sensor network layer, in which large numbers of low-cost battery-powered motes are deployed in the environment to perform sensing, processing and communication tasks. Open-source software such as TinyOS allows cross-platform compatibility of motes with different processing capabilities and sensor interfaces, and provides a common communications interface over low-power 802.15.4 PAN radio. Mote-level devices are essentially designed for low duty-cycle operation and required to periodically sense the environment at intervals, or to respond to queries initiated by a higher-level device such as a cluster-head node. As mentioned in our two-tier cluster architecture in section (3.5.1), each *logical* cluster of WSN motes can be managed by a WSN gateway such as a Stargate.

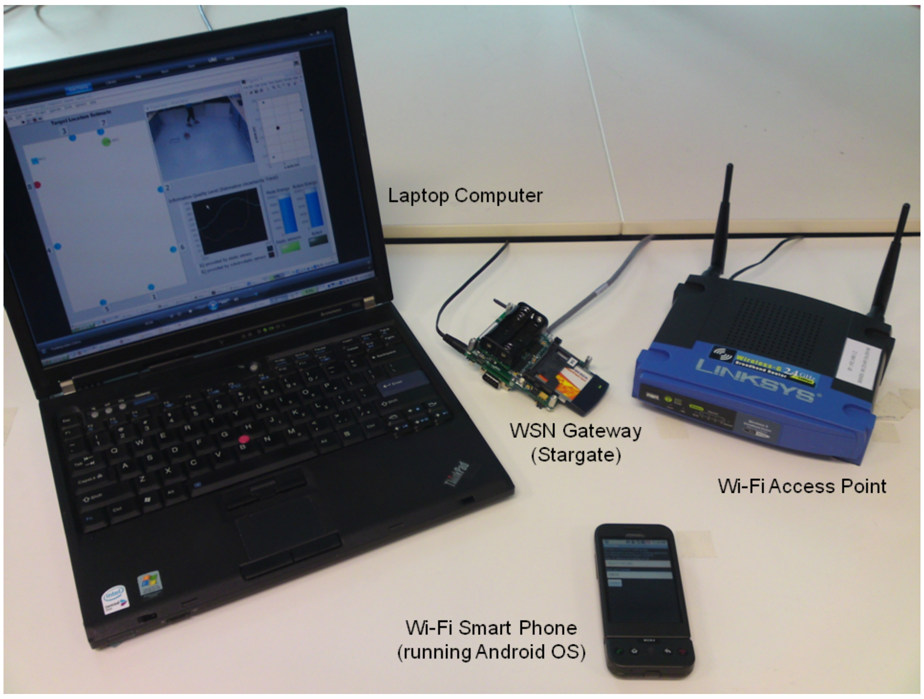


Figure 3.14: Mobile Devices connected by Wi-Fi ad-hoc network

The middle layer consists of mobile devices supported by popular ad-hoc wireless network technologies, such as 802.11 Wi-Fi. Devices that fall in this category include computer-class devices (laptops, netbooks), mobile platforms (PDAs, tablets) and smart-phones (e.g. iPhone, Android-based phones). These devices can communicate with one another as peers in an ad-hoc Wi-Fi network, as well as with WSN Stargate gateways, each attached with a CompactFlash Wi-Fi card. This allows mobile devices to access real-time information from the WSN, such as sensor readings, battery level, link quality and network topology.

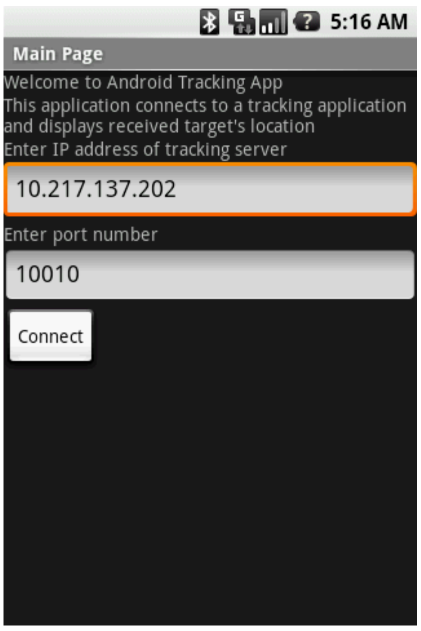
Figure 3.14 shows a few of the mobile devices which were used to implement the proposed architecture in our test-bed.

The top layer in the proposed architecture consists of the software modules running on the mobile devices, for which multiple modules can exist on the same device. For example, an Android application can contain modules for WSN monitoring (logging sensor data), sensor querying (getting location coordinates) and visualization (displaying real-time location on-screen). Modules communicate with one another over socket interfaces – a client application running on a mobile phone may connect to a remote WSN application, to obtain streams of sensor data of temperature and lighting measurements of a room in another building. More abstract communication structures can be built over these socket interfaces, such as SQL-like queries which are suitable for aggregation of sensor data across multiple physically co-located but logically distinct WSN clusters.

For example, a data aggregation and visualization module may request information from a querying application using an SQL-like query: SELECT readings FROM house WHERE value > THRESHOLD. The querying module parses such information into sub-queries to decide the querying procedure, e.g. which sensors to query, and in what order, before sending the sub-queries to the WSN gateway, which converts them into the proper packet format to obtain the respective sensor measurements. On the return path, data can be aggregated and/or grouped into node clusters to conserve communication resources, and the querying module converts the raw data into a form suitable for the visualization module's interface. Queries may also make use of context information, for example to increase the sampling rates of ambient sensors when an elderly person is detected in a smart room.

3.6.4 Tracking Application on an Android Smartphone

We made use of the software components and interfaces in the Android SDK to extend our indoor target tracking test-bed implementation on a HTC smart-


5:16 AM

Main Page

Welcome to Android Tracking App
This application connects to a tracking application and displays received target's location
Enter IP address of tracking server

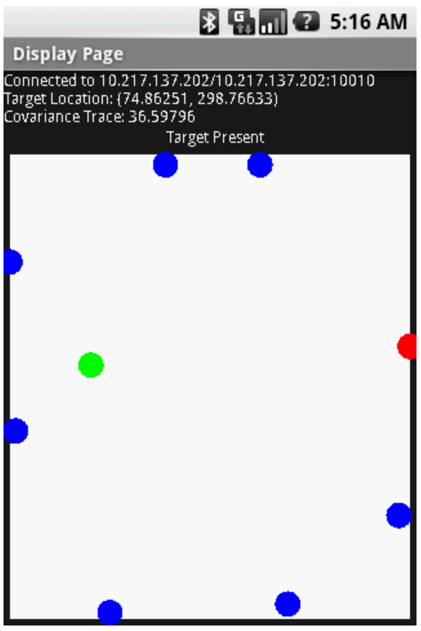
10.217.137.202

Enter port number

10010

Connect

(a) GUI to Connect to Tracking Server


5:16 AM

Display Page

Connected to 10.217.137.202/10.217.137.202:10010
Target Location: (74.86251, 298.76633)
Covariance Trace: 36.59796

Target Present

(b) Tracking Visualization

Figure 3.15: Android Tracking Visualization Application

phone. Specifically, we developed an Android location visualization module and used it to communicate with the indoor tracking application server in the test-bed. The monitoring module connected to the tracking server over a TCP socket interface (Figure 3.15(a)), and parsed the returned data to obtain the following information:

- Person Presence/Absence
- Location coordinates (x, y)
- Measure of uncertainty (Covariance Trace)
- Sensor Selected

The Android visualization module (Figure 3.15(b)) displayed the real-time information and animated the received target coordinates, providing the mobile phone user with real-time target location updates. Such a system can allow for remote monitoring of elderly in smart homes, or for indoor self-localization applications in museums and galleries.

3.7 Discussions

Our test-bed prototype was developed in order to study and implement algorithms for indoor target tracking and resource management for ambient sensing applications. We made use of relatively low-cost commercially-available off-the-shelf (COTS) devices, such as ultrasonic sensors and wireless sensor network mote platforms. As a result, our challenge was to develop a tracking application under significant resource constraints. In this section, we discuss the limitations of our test-bed implementation and we identify some possible extensions for more general application settings [57].

Despite these limitations, our approach to focus on ambient sensors instead of RFID-tag based solutions addresses the requirements of certain monitoring applications which require minimum interaction with the target being tracked, such as healthcare-related activity monitoring applications for the elderly at home, or intrusion detection and tracking systems. In addition, we did not make use of image or video cameras as these raise privacy concerns.

3.7.1 Limitations and Challenges

Sensors

We were significantly constrained by the ultrasonic sensors we used, in terms of fidelity and coverage. As ultrasonic sensors obtain range measurements via time-of-arrival of reflected ultrasonic pulses, they are vulnerable to obstacles in the detection region. Using ultrasonic sensors in the context of an indoor monitoring application poses significant problems in obtaining accurate measurements, due to reflections off furniture and other object clutter. In addition, due to the range measurement mechanism, multiple ultrasonic sensors with overlapping sensing regions cannot operate at the same time, as some of them would pick up reflected pulses from one another, resulting in inaccurate measurements. Due to a minimum sensing time required to aggregate across all received pulses in order to decide on the range measurement, the sensors have to be scheduled and carefully calibrated to provide sufficient sensing coverage, as well as to avoid inter-sensor interference. In addition, it is very challenging to make use of ultrasonic sensors as the only sensing modality for multiple target tracking applications.

Data Processing

To address the resource constraints posed by low-power mote devices, we tried to distribute the processing load across sensor nodes so that the target tracking could be carried out in a distributed and self-organised manner. The target state estimate would be maintained and updated at the currently-active sensor node, and passed to the next-best sensor node according to the information-quality (IQ) metric described earlier in this chapter. However, in our implementation, we found that even small packet loss rates at around 5% could result in severe degradation in the state estimate, as the entire state was lost and it would take a long time for the system to reinitialise to track the target state, often resulting in a re-initialization of the EKF algorithm.

In addition, packet loss increased with the transmission of larger packets containing the entire EKF state estimate, and even a few intervals of missed observations, at a sampling interval of 100ms, would also cause large discontinuities in the trajectory estimate, greatly affecting tracking accuracy. As such, we kept to a centralized implementation of the tracking algorithm, in which sensor nodes simply performed sensing and forwarded small packets containing only measurement information to the data sink, which was implemented on a more-powerful Stargate processing platform, within a one-hop star network topology. As the application was constrained by sensing and processing speeds and a small network topology, medium access contention issues were not significant enough to cause degradation of application performance.

Computational Algorithms

As our application was developed on low-power wireless sensor network motes, the processing limitations significantly affected the type of algorithms we could implement. We chose the Extended Kalman Filter (EKF) as it was relatively simple to implement. However, EKF is known for its vulnerability to non-linear motion dymanics [58], suffering from large convergence problems even when the dynamics are only slightly non-linear. Linearization using Jacobian partial derivatives served as an approximation approach, which may not work well depending on how erratic or random the target trajectory is. In addition, it is challenging to tune the EKF to accurately capture the process and observation noise model parameters and the covariance matrices.

In target tracking applications in which the sensors used are able to sample at sufficiently-high speeds, the target motion within a sensing interval may be assumed to be almost linear. However, given the sensing and processing constraints of our low-cost hardware platforms, our EKF algorithm would only be able to track relatively simple motion models, such as the circular and rectangular target trajectories shown in our earlier simulations. More random motion models may result in mis-predictions of the EKF algorithm, resulting in target misses and large discontinuities in the trajectory estimate as the EKF tries to recover the target.

3.7.2 Extensions

Multi-Modality Sensing

One approach to extend our test-bed implementation to more general application settings would be to add a variety of other sensor modalities, such as passive infra-red motion detectors, microphone arrays and pressure sensors, so that the different coverage regions of different sensing modalities could help to reduce the uncertainty in the target's state location estimate. However, the EKF algorithm would only apply to sensor observation models with Gaussian noise distributions. Non-Gaussian noise models could make use of a mixture of Gaussian models [58], or a particle filter.

The particle filter [25] is a Monte-Carlo simulation approach to represent the distribution of the state estimate by a collection of particles, each representing a possible state estimate. Particles are associated with individual weights, which represent the relative importance of each particle to the entire distribution of particles. A simple approach to aggregate the contribution of all the particles to the overall state estimate could be to take the weighted sum of particles. This approach can take into consideration the contributions of sensors of different modalities and non-Gaussian noise, as well as non-linear process models.

However, the computational complexity is significantly increased, as it is difficult to decide how to prune the distribution of particles to remove those which are deemed to have little contribution, how to represent the likelihood function which determines which observations to associate with which particles, as well as what measure of the particle distribution to use as the updated state estimate. In addition, it is much less straight-forward to design and update an information-quality metric for a particle filter and the corresponding utility metric. Last but not least, the computational resources required to store and process a large number of particles in order to have a reasonably accurate estimate would be quite substantial and the processing platforms would have to be more powerful that the wireless mote platforms currently available.

Multi-Target Tracking and Mobile Sensing

In an extension of our test-bed implementation, our indoor tracking application was complemented with a mobile robot with an on-board laser scanner for high-resolution sensing. The laser provided more accurate measurements at higher resolutions and sensing rates, thus serving as a more powerful sensor that could greatly increase the information gain in the tracking algorithm. However, this also introduced many new challenges, such as data association of laser scan readings to differentiate between furniture, walls and the two legs of the human target.

In addition, the robot with the laser scanner consumed energy at a much higher rate that the ambient ultrasonic sensors and wireless motes, thus limiting its effective operation periods. We made use of the robot to improve on the tracking accuracy when the EKF algorithm using the ultrasonic sensors lost the human target and needed to quickly re-capture the location estimate. The resource management problem thus focused on deciding whether to trigger the robot or the sensors, and where to move the robot to obtain a good measurement. We note that despite its computation and data association challenges, the laser scanner modality would be one of the most suitable sensors to extend to multiple target tracking.

3.8 Conclusion

This chapter has presented the implementation work in our test-bed. We have designed and implemented a WSN test-bed for target tracking, and a two-tier cluster architecture for resource management. We have also proposed and implemented a software architecture for integrating smartphones with WSNs and other mobile devices, such as netbooks and PDAs, over Wi-Fi. Using the Android SDK, we have developed software components for the monitoring, control and visualization of WSN applications, which we have applied to our indoor target tracking test-bed implementation, to achieve real-time remote monitoring and visualization on mobile devices. We have also identified the limitations and challenges of our implementation, and discussed suitable extensions towards a more general application setting.

Chapter 4

Information-driven Sensor Election and Routing

4.1 Introduction

This chapter describes an information-driven and energy-aware approach for distributed sensor election and multi-hop routing in wireless sensor networks. The adaptive sensor selection scheme for target tracking in a single-hop sensor network from Chapter 3 is extended to a distributed sensor election scheme in a multi-hop sensor network. In Chapter 3, the current sensor selected the next node by computing the expected information gain of each candidate sensor. In this chapter, our distributed sensor election mechanism lets candidate nodes elect themselves for sensing tasks to update the estimation algorithm at different intervals, in order to conserve energy while keeping within constraints of the information quality (IQ) metric.

After the current sensor node updates the state estimate, the state information is propagated along multiple hops back to the sink node, based on the remaining energy of neighboring nodes. Next-hop nodes are chosen to minimise the sum of expected costs towards the sink, in order to achieve the objective of increasing network lifetime. Reinforcement learning is used for nodes to learn which of their neighboring nodes to forward to, so as to conserve energy and ensure delivery to the sink node.

4.2 Related Work

4.2.1 Competition-based Sensor Selection

In [1], a distributed sensor selection scheme was implemented in an indoor human target tracking test-bed, in which the active node broadcasted its updated state estimate to its neighbors, which distributively computed their expected information gain based on their locations. Sparse matrix computation was used to reduce the computational time, and candidate nodes elected themselves as the next active node using a back-off scheme, such that their back-off times were inversely proportional to their respective amounts of expected information gain. As a result, the node with the most information gain replied first, so that distributed computation using sparse matrices reduced the computation time, and hence the update intervals for the estimation filter, and message exchange was reduced by the election mechanism. However, [1] did not consider data routing back to the sink node.

4.2.2 Multi-step Look-ahead for Data Routing

Based on information-utility metrics formulated in [5], and sensor querying and data routing approaches in [19], the authors of [59] formulated the routing problem as joint optimization of data transport and information aggregation. A Bayesian inference framework was used to represent and update the belief state, and mutual information was used as a metric to characterize information gain. The IDSQ and CADR approaches used in [19] were used for information routing under two different scenarios: (i) routing a query to a region of rich information about the state estimate using information-directed multiple-step look-ahead to avoid sensor holes, with a min-hop algorithm that improved upon [19] in tracking performance, and (ii) routing to a designated exit node, in which a real-time extension of A* heuristic search was used as a forward search algorithm to route from query source to designated exit node, such that the routing path was attracted to the information region, thus increasing the tracking performance significantly.

4.2.3 Routing with Reinforcement Learning

In [60], a straight-forward application of Q-learning was presented for packet routing, to discover routing policies that minimise path lengths without knowing the network topology or traffic patterns in advance, and without a centralized routing control system. A large table of Q-values was used to represent expected costs, in this case, source-destination routing delays. Q-routing was shown to out-perform statically computed shortest-paths obtained by the Bellman-Ford algorithm, and adapt to changing network topologies, traffic patterns and load levels. The authors highlighted the problem of Q-routing being unable to recover from erroneous estimates, and a 'full-echo' Q-routing approach was attempted to address this issue, in which neighbors were queried on their Q-values before the routing decisions were made. However, results were worse than the basic Q-routing scheme under high-load, as the 'full-echo' Q-routing seemed to be constantly changing and oscillating between policies, resulting in unstable behavior and worse performance.

Routing with reinforcement learning was also presented in [61] in which three meta-heuristic algorithms were used to perform routing within a reinforcement learning framework. Each node maintained a Q-value, which was defined to be the minimum cost from that node to the destination. Since the destination node was mobile, nodes had to discover and improve existing routes in an online fashion. Three meta-strategies were used: (i) *real-time search* which tried to find the best neighbor, (ii)*constrained flooding* which was used to decide if and when to re-broadcast packets, and (iii) *adaptive spanning tree*[62] which decided which parent node to forward packets to.

In [63], the approaches in [59] were extended, based on the limitations that the M-hop look-ahead decision horizon was computationally expensive with significant message exchange needed, and the real-time A* heuristic search assumed that the exit node was known, which were not applicable for applications in which the destination was not known apriori, or for a mobile destination node. From [61], the reinforcement learning approach to multi-hop routing was adopted to formulate a weighted shortest-path problem with an additive objective metric. Initial Q-values were sent by flooding from the destination node, and each node maintained estimates of its neighbors' Q-values. Routing paths were generated by considering both communication-based metrics, such as small hop-count, and information-based metrics, such as tracking accuracy, even with gaps in network connectivity and unpredictable moving destination nodes.

4.3 Our Proposed Approach

Our proposed approach consists of two phases, a distributed sensor election phase and a multi-hop routing phase. Similar to the tracking scenario in Chapter 3, sensor nodes stay in an idle state until a target is detected by one of the nodes, which initializes the state estimate and invokes the tracking algorithm to perform tracking and distributed sensor election. However, while Chapter 3 addresses centralized sensor selection within a single-hop sensor network, this chapter addresses distributed mechanisms within a multi-hop sensor network.

Our distributed sensor election mechanism is motivated by [1], in which the current node performs sensing and data fusion to update the current state estimate, and broadcasts it to neighboring nodes for them to elect themselves as the next sensor node with a back-off delay. The contribution of the competition-based sensor election mechanism in [1] was to use distributed processing to reduce the computation time for the current node to find the best sensor node. Hence, the sensing interval, which is taken to be the time between the broadcast of the prior state until the first reply from a candidate node, is reduced. As a result, the tracking error is also reduced. After handover to the next sensor node, it is assumed to perform sensing immediately.

We adopt this approach to allow nodes to update the tracking algorithm with a dynamic sensing interval based on their *Information Quality* (IQ) metric, in contrast to Chapter 3, in which nodes were selected based on a static sensing interval for computing the prior state estimate in the *predict* phase. In addition, each candidate node is allowed to trade-off its expected information gain and its remaining energy in deciding its back-off time. Furthermore, the node that wins the sensor election process can introduce additional sensing delay, depending on its IQ metric, so as to further conserve its remaining energy, or that of neighboring sensor nodes. The detailed mechanism is described in Section 4.4.

After the distributed sensor election phase, the current node forwards its updated state estimate back to the sink node via multi-hop routing. Unlike AODV in which explicit routes are initialized and subsequently maintained in the event of route failure, our approach allows nodes to discover routes to the sink node and iteratively improve on them using a feedback signal that represents the utility of their forwarding decisions, similar to that in [61].

In similar work in Information-Directed Sensor Querying (IDSQ) [59], the order of querying nodes is defined using an information-utility metric, and Constrained Anisotropic Diffusion Routing (CADR) subsequently makes use of an objective function to trade-off information gain and remaining energy of nodes, for queried nodes to make routing decisions to propagate their measurements and state estimates to a sink node. Our approach separates the sensor selection and routing processes, in which the sensor election phase focuses on the next-hop neighborhood to select nodes based on the changing event (the moving target). Subsequently, the energy-aware multi-hop routing phase makes use of the remaining energy and cost metrics of one-hop neighboring nodes. The order of nodes to be queried may not be computed in advance as the target motion is unknown. Only the next-hop neighboring sensor node for the next timestep is known, after it wins the sensor election process based on its information gain and remaining energy.

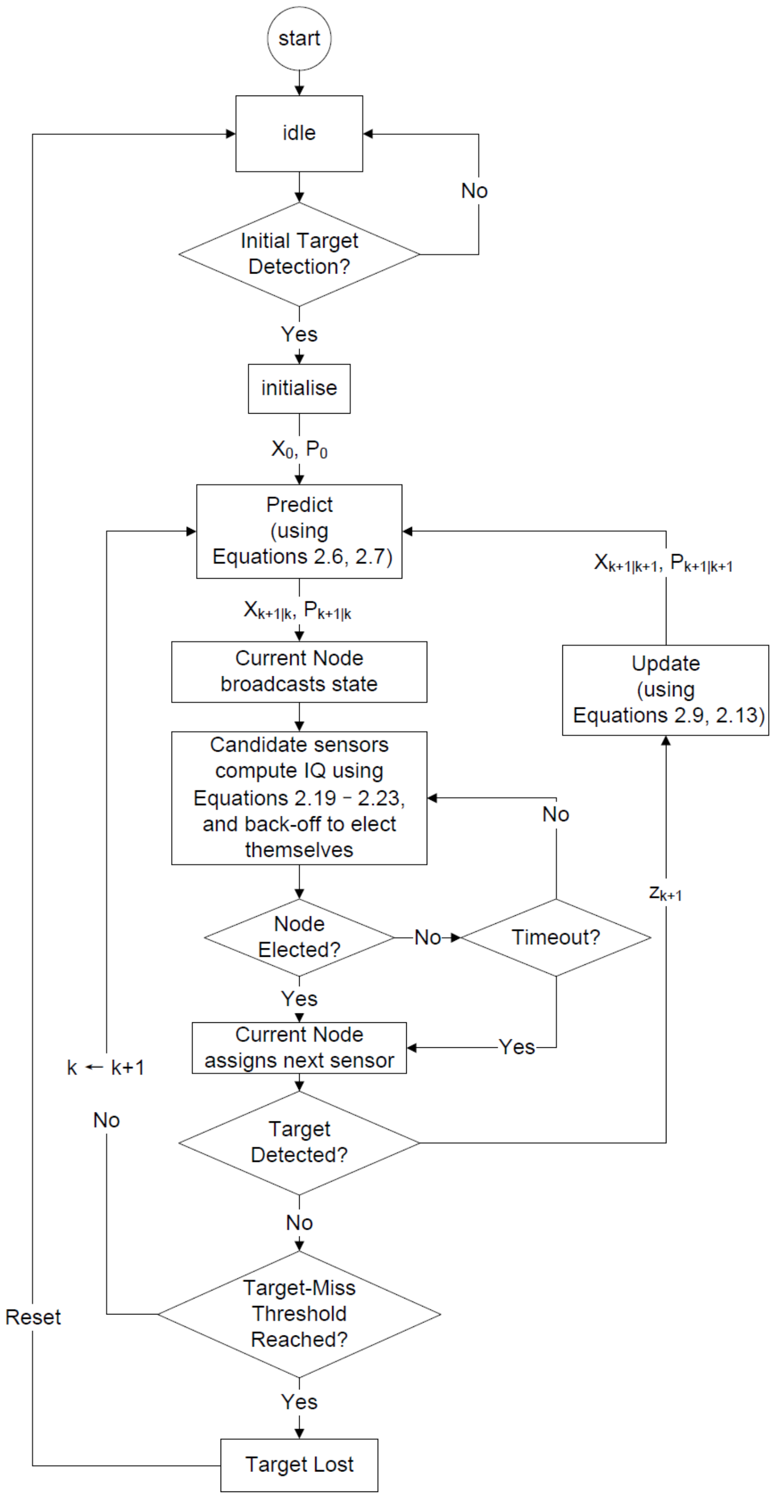
At each timestep, the sensing node serves as the source node, and it forwards the updated state estimate by choosing the next-hop neighbouring node with the minimum expected sum of costs to the sink node. We use a straightforward application of Q-learning to discover energy-efficient routes to the sink node that avoid energy-depleted nodes, and strive to improve on sensor network lifetime and delivery ratio while minimising tracking error. The routing objective differs from the mechanism in Message Constraint-based Routing in [61], which attempts to route towards a mobile destination and avoid sensor holes, but our reinforcement learning approach is similar. The detailed mechanism for energy-aware multi-hop routing will be described in Section 4.5.

4.4 Distributed Sensor Election based on Information Gain and Remaining Energy

4.4.1 Distributed Sensor Election Mechanism

Figure 4.1 shows a flowchart with the EKF and sensor selection algorithms described in Chapter 2. Similar to Figure 3.4 from Chapter 3, the recursive *predict-update* mechanism of the EKF is used to keep track of the state estimate of the target location. The process model for the *predict* phase is represented by the matrix $F(\Delta t_k)$, to provide the prior state estimate, where Δt_k is the sensing interval at step k. After a measurement is taken, the *update* phase computes the posterior state estimate, and the candidate sensors for the next timestep are selected based on their information quality, $IQ_i = trace(\widehat{P}_{i,k+1|k+1})$, a cost metric which represents the estimation uncertainty in the EKF algorithm for candidate node i.

The main contrast with Figure 3.4 from Chapter 3 lies in the sensor selection phase. Instead of the current node computing the expected IQ metrics of



start
idle
Initial Target Detection?
No
Yes
initialise
X_0, P_0
Predict (using Equations 2.6, 2.7)
$X_{k+1|k}$, $P_{k+1|k}$
Current Node broadcasts state
Candidate sensors compute IQ using Equations 2.19 - 2.23, and back-off to elect themselves
Node Elected?
No
Timeout?
No
Yes
Yes
Current Node assigns next sensor
Target Detected?
z_{k+1}
Update (using Equations 2.9, 2.13)
$X_{k+1|k+1}$, $P_{k+1|k+1}$
No
Target-Miss Threshold Reached?
No
$k \leftarrow k+1$
Yes
Target Lost
Reset

Figure 4.1: Flowchart for State Estimation and Distributed Sensor Election

each candidate node using a static sensing interval Δt_k, the current node now broadcasts its updated next-step prior state estimate after the predict phase, and candidate nodes compute their respective IQ metrics in a distributed manner. For a candidate node i, it computes IQ_i using the prior state estimate and its predicted measurement, and it evaluates a cost function that includes its remaining energy e_i, as shown in Equation 4.1.

$$cost_i = \beta \frac{trace(\widehat{P}_{i,k+1|k+1})}{trace(\widehat{P}_{k+1|k})} - (1-\beta)\frac{e_i}{e_{max}} \tag{4.1}$$

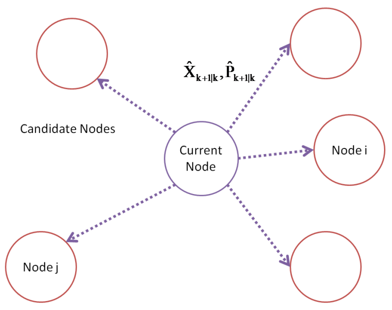
Here, $trace(\widehat{P}_{k+1|k})$ represents the trace of the prior state covariance matrix, based on the *predict* phase in the EKF algorithm, as shown in Equation 2.7, with sensing interval Δt_k set to be a constant value, ΔT, which represents the maximum timeout for sensor election. The quantity $trace(\widehat{P}_{i,k+1|k+1})$ represents the trace of the predicted posterior state covariance matrix, given the predicted measurement $z_{i,k}$ of candidate node i. The variable e_i represents the remaining energy level of node i, and e_{max} is its initial energy level.

From Equation 4.1, the covariance trace and remaining energy of node i are normalized to reflect a suitable scale for comparison and trade-off. Since Equation 4.1 represents a cost function and $trace(\widehat{P}_{i,k+1|k+1})$ represents a measure of uncertainty, the remaining energy e_i is assigned a negative coefficient as it represents a utility value. The parameter β reflects the relative weight of the IQ ratio to the energy ratio in the composite cost function. An increasing value of β indicates increasing priority given to the IQ cost component.

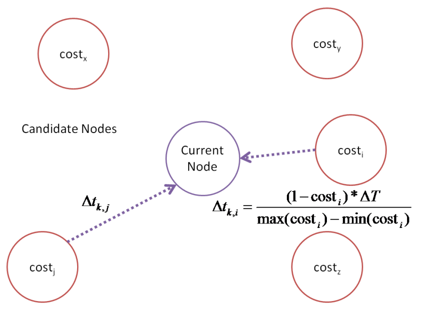
Due to the normalization components in Equation 4.1, the range of values of the cost function $cost_i$ is $[-1, 1]$. Each candidate node i makes use of $cost_i$ to select its back-off interval $\Delta t_{k,i}$, subject to a timeout threshold value, ΔT, as shown in Equation 4.2.

$$\Delta t_{k,i} = \frac{(1 - cost_i) * \Delta T}{max(cost_i) - min(cost_i)} \tag{4.2}$$

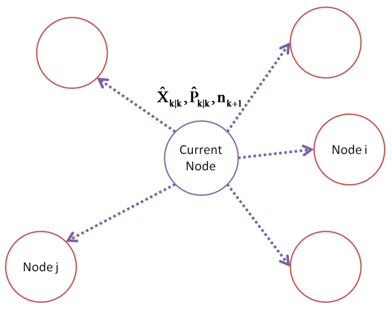
Here, $max(cost_i)$ and $min(cost_i)$ represent the maximum and minimum values of $cost_i$, which are 1 and -1 respectively. Equation 4.2 translates $cost_i$ to the range $[0, 2]$, normalizes it within the range of values for $cost_i$, and multiplies it by the timeout threshold, so that each candidate node i can determine its back-off delay based on its IQ and remaining energy, subject to the timeout threshold value. The higher the cost of a node, the more its back-off delay in the distributed sensor election procedure.



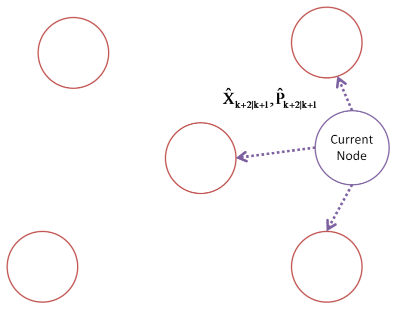
(a) Current node at timestep k broadcasts prior estimate $\widehat{X}_{k+1|k}, \widehat{P}_{k+1|k}$



(b) Each candidate node i computes $cost_i$ and back-off delay $\Delta t_{k,i}$



(c) Current node assigns current state estimate $\widehat{X}_{k|k}, \widehat{P}_{k|k}$ to winning node i



(d) Node i updates $\widehat{X}_{k+1|k+1}, \widehat{P}_{k+1|k+1}$ and broadcasts $\widehat{X}_{k+2|k+1}, \widehat{P}_{k+2|k+1}$

Figure 4.2: Distributed Sensor Election Procedure

Figure 4.2 shows the message exchange protocol for our distributed sensor election procedure. At timestep k, the current node uses a fixed timeout threshold value ΔT of 100ms as the sensing interval, in order to compute the prior state estimate $\widehat{X}_{k+1|k}, \widehat{P}_{k+1|k}$ and broadcast it to its one-hop neighboring nodes, as shown in Figure 4.2(a). Each candidate node i that is able to detect the target computes $cost_i$ and replies after a backoff delay of $\Delta t_{k,i}$, as shown in Figure 4.2(b). Nodes which have lower costs are more valuable, so they use a short back-off delay to elect themselves earlier. Upon hearing a node's response, other candidate nodes refrain from sending their responses.

Figure 4.2(b) also illustrates the *hidden node* problem, in which nodes on the opposite side of the current node, such as node j, may be unable to overhear node i's response. As shown in Figure 4.2(c), the current node broadcasts the winning node's identity, together with the current state estimate $\widehat{X}_{k|k}, \widehat{P}_{k|k}$, so that candidate nodes are aware of the successfully elected node. This mechanism is also used to enforce sensor selection when no response is received by the election phase timeout, for which one node is randomly chosen from the set of neighboring nodes.

Upon receiving its assignment, node i computes the actual prior estimate $\widehat{X}_{k+1|k}, \widehat{P}_{k+1|k}$ using its sensing delay $\Delta t_{k,i} \leq \Delta T$, takes its measurement z_{k+1}, and updates the posterior state estimate $\widehat{X}_{k+1|k+1}, \widehat{P}_{k+1|k+1}$. Subsequently, it uses ΔT as the sensing interval to predict the next prior state estimate $\widehat{X}_{k+2|k+1}, \widehat{P}_{k+2|k+1}$, and broadcasts it to its one-hop neighboring nodes, repeating the sensor election process, as shown in Figure 4.2(d).

4.4.2 Delayed Sensing based on IQ Metric

From the previous section, node i with the least back-off delay wins the distributed sensor election procedure, based on its composite cost function that trades-off IQ and remaining energy, as it is the most cost-effective node to perform sensing in the next timestep. This section describes the idea of further extending the sampling interval by allowing node i to delay taking a measurement based on its IQ metric. The rationale is that if node i is able to provide a large value of information gain, it can delay taking a measurement to allow the uncertainty to increase to a threshold value. The increased sampling interval helps to conserve the remaining energy of node i and next-hop nodes, subject to a constraint on the IQ metric.

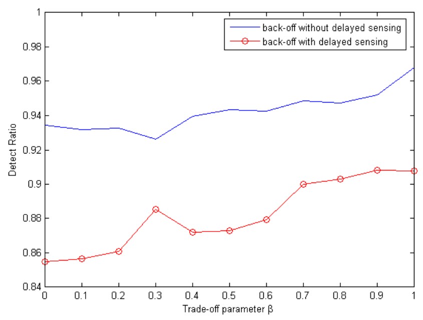
$$delay_i = \frac{trace(\widehat{P}_{k+1|k}) - trace(\widehat{P}_{i,k+1|k+1})}{trace(\widehat{P}_{k+1|k})} * timeout \tag{4.3}$$

Equation 4.3 shows the amount of *additional* sensing delay that node i can afford. The difference between node i's expected IQ, given by $trace(\widehat{P}_{i,k+1|k+1})$ with sensing interval $\Delta t_{k,i}$, and the estimation uncertainty in the updated prior state estimate, $trace(\widehat{P}_{k+1|k})$ with sensing interval ΔT, is normalized by $trace(\widehat{P}_{k+1|k})$ and multiplied with the timeout threshold value. Lower values of the expected estimation uncertainty of node i, given by a lower $trace(\widehat{P}_{i,k+1|k+1})$ value, will result larger values of additional sensing delay, subject to the timeout threshold value.

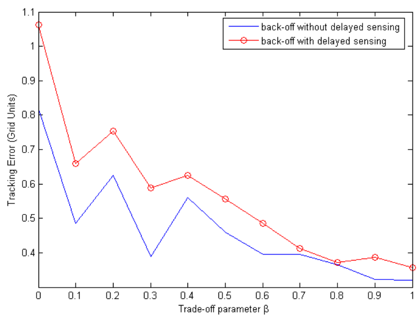
4.4.3 Simulation Results

Simulations were conducted for the distributed sensor election procedure described in Section 4.4.1 to compare the effects of adding the delayed sensing mechanism described in Section 4.4.2. The sensor network configuration was a grid of 10x10 units, in which node locations were slightly perturbed from the grid points with uniformly distributed noise. The target was assumed to move in a circular trajectory of radius of four grid units, and one round around the circular trajectory was regarded as one tracking cycle.

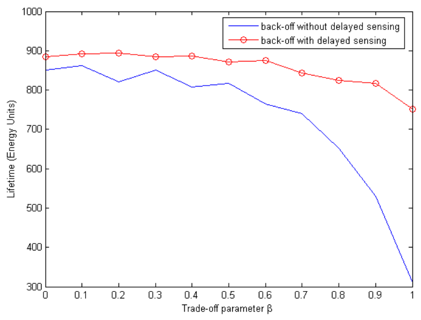
Fixed values of sensing and communication energy were used, and simulations were conducted for 10 cycles for two scenarios each: (i)distributed sensor election only, and (ii) distributed sensor election with delayed sensing. The back-off delay in the distributed sensor election procedure is given by Equation 4.2 and simulation were conducted for values of the trade-off parameter β from 0 to 1.0 in increasing steps of 0.1. Figure 4.3 shows our simulation results for the distributed sensor election procedure, comparing its performance with and without our delayed sensing mechanism, in terms of target detection ratio, tracking error and lifetime.



(a) Plot of ratio of target detections



(b) Plot of tracking error in grid units



(c) Plot of sensor network lifetime

Figure 4.3: Simulation results for distributed sensor election with and without delayed sensing

As β increases from 0 to 1.0, increasing emphasis is placed on the cost of the IQ metric, so the sensor election mechanism puts an increasing weightage on reducing estimation uncertainty. As expected, Figure 4.3(a) shows that the detection ratio increases with β for both curves, and Figure 4.3(b) shows that the tracking error is reduced as β is increased. The improved tracking performance comes at a cost of sensor network lifetime, as shown in Figure 4.3(c). Due to increased emphasis on IQ in the composite cost function in Equation 4.1, sensor election favors nodes with higher IQ, at the expense of their remaining energy.

In addition, Figure 4.3 show that with delayed sensing, the target detection ratio is slightly decreased and tracking error is slightly increased, but the sensor network lifetime is greatly increased, especially at larger values of β. Although increasing the value of β places increasing emphasis on IQ in the composite cost function (Equation 4.1), the *lowest-cost* candidate node has sufficiently low IQ cost to afford to delay its sensing interval significantly. Using the distributed sensor election procedure together with delayed sensing allows for energy conservation, without sacrificing too much of the tracking performance.

4.5 Energy-Aware Multi-Hop Routing

In this section, we describe our approach to forward the state estimate from the sensing (source) node via multi-hop routing to the sink node, after the distributed sensor election procedure determines the source node at each timestep. The source node changes according to the unpredictable target motion, and subsequently, energy-aware multi-hop routing is performed to forward the updated state estimate back to the sink node, independent of the sensor election procedure.

At each hop in the multi-hop routing process, the current node could make a forwarding decision based on the remaining energy levels of its neighbour nodes. However, this may result in packet forwarding to a promising neighbour node with high remaining energy, only to subsequently encounter a region of energy-depleted next-hop nodes, which would reduce the network lifetime further.

We use reinforcement learning for nodes to discover and maintain routes to the sink node. The remaining energy of each node i can be converted into a cost metric c_i using an energy-aware cost function $c_i = c(e_i)$, where e_i indicates the current node's remaining energy level. Based on this cost function, energy-aware routing from source to sink node is converted into a minimum-cost routing problem. Based on its remaining energy level, each node i maintains its cost metric c_i, as well as an estimate of its *distance* metric, d_i, which is an expected sum of costs to the sink node. The distance metric summarises the expected future costs of forwarding packets to the sink node, in terms of hop count, as well as the remaining energy level. As a result, using the distance metric to select the next-hop node can help to avoid regions of energy-depleted nodes.

Our proposed approach for multi-hop routing is similar to distance-vector routing [64], in which link costs can be used to represent propagation delays due to transmission distance, or other cost-based metrics. We express the cost metric as an inverse function of remaining energy to balance the nodes' energy consumption. Nodes which are often chosen to forward packets will find their cost and distance metrics increasing quickly, thus prompting their parent nodes to select an alternative node to forward to. In our approach, cost and distance metric values change rapidly within a few packets exchanged, so in order to reduce message exchange and excessive computation in resource-constrained sensor nodes, only local information, in the form of cost feedback from one-hop neighbors, is used to update each node's cost and distance metrics.

4.5.1 Problem Formulation

This section describes the system models for energy-aware multi-hop routing, which is modelled by a Markov Decision Process.

State

The state of a node i is its remaining energy level e_i, which lies between 0 and e_{max} and is partitioned into E energy levels.

Action

The actions available to a node i correspond to the links to neighboring nodes that it can transmit to. We assume that such information is provided to each node by a neighbor discovery protocol, in which nodes which lie within a fixed communication range are assigned to be neighbours of one another. We do not assume topology changes due to changes in transmission power.

Cost Function

For each node i, its cost metric c_i is given by a function $c(e_i)$, with the requirement that $c(e_1) \leq c(e_2)$ if $e_1 \geq e_2$, i.e. the less the node's remaining energy level, the more costly it is to forward to that node. We make use of a cost function $c(e_i) = e_{max}/(e_i + 1)$, where the cost of a node c_i is inversely-related to its remaining energy level e_i, and e_{max} is the maximum energy level.

Value Function

For each node, we denote the *Q-value* of each state-action pair, $Q(s, a)$, to represent the *cost-to-go* from itself to the sink node, given its current state $s_i = e_i$ (remaining energy level), in which it selects an action $a_i \in A(s_i)$ (the neighboring node to forward to). For a node i, its distance metric, d_i, represents its minimum expected sum of costs to the sink node, which is taken to be the minimum Q-value for all possible actions in the current state:

$$d_i(s) = min_{a_i} Q_i(s_i, a_i) \tag{4.4}$$

4.6 Solution by Reinforcement Learning

4.6.1 Solution Approach

The Q-value of each state-action pair of each node depends on its energy level, the energy level of its neighbors and its location relative to the sink node. In order to compute the optimum Q-values for all possible state-action values at each node, the number of messages exchanged between nodes to communicate such values would consume significant overhead.

Hence, we make use of reinforcement learning to let the nodes learn their Q-values, based on cost feedback information from their next-hop neighboring nodes. In order to perform exploration to find potentially better solutions, nodes need to adopt an ϵ-greedy policy [36] to forward to a random neighboring node (random action) with probability ϵ and to the node with the minimum Q-value (greedy action) with probability $1 - \epsilon$, i.e.

$$\pi(s) = arg\ min_{a_i} Q(s_i, a_i) \tag{4.5}$$

Nodes learn the utility of their actions using cost feedback from their next-hop neighbors, which is subsequently used to update their Q-values:

$$Q(s_i, a_i) \leftarrow (1-\alpha)Q(s_i, a_i) + \alpha * [c_i + min_{a_i} \gamma Q(s'_i, a'_i)], \tag{4.6}$$

where α represents the learning rate and γ represents the discount rate, which indicates how much a future cost is valued at the current step. We adopt the offline Q-learning approach [36] to update with the minimum Q-value among the next state-action pairs $Q(s'_i, a'_i)$. Note that Equation 4.6 shows that the update to $Q(s_i, a_i)$ incorporates the previous value and the *temporal-difference* between the received cost c_i and the predicted next $Q(s'_i, a'_i)$ value, which is damped by the learning rate α.

4.6.2 Solution Algorithm

In this section, we describe our reinforcement learning-based algorithm for energy-aware multi-hop routing, which consists of two phases: an *initialisation* phase and a *tracking and forwarding* phase. Packets are broadcasted with the format $< destination\_node, current\_cost\_metric, current\_distance\_metric >$. Node i's message would be of the form $< n_i, c_i, d_i >$, in which n_i depicts the destination node that node i is sending to, and c_i and d_i represent node i's cost and distance metrics respectively. The algorithm pseudo-code is shown in the next page.

Initialisation Phase

In the initialisation phase, nodes initialise their cost metrics based on their initial energy levels $c_i = c(e_i)$, and set their Q-values and distance metrics d_i to ∞. The sink node starts broadcasting its cost and distance metrics set to zero: $< BROADCAST, 0, 0 >$.

One-hop neighbour nodes who overhear this message update their Q-values. If the corresponding $Q(s, a)$ is set to ∞, which indicates that this is the first time

Algorithm 5: Energy-Aware Multi-Hop Q-Routing

```
**Initialisation phase:**
for $all\ nodes\ i \in N$ do
    $Q_i(s,a) \leftarrow \infty, \forall s \in S, a \in A$
    $c_i \leftarrow e_{max}/(e_i+1), d_i(s) \leftarrow \infty$
end
$Q_d(s,a) \leftarrow 0, \forall s \in S, a \in A$ for destination node $d$,
$c_d \leftarrow 0, d_d(s) \leftarrow 0$
broadcast $\langle n_0, c_d, d_d(s)\rangle$
while $initialisation\_incomplete$ do
    received $\langle n_0, c_j, d_j\rangle$ at $i$ from node $j$ at link $a_i$
    $\delta \leftarrow min_a Q_i(s,a)$
    if $Q_i(s,a_i) = \infty$ then
        $Q_i(s,a_i) \leftarrow c_j + d_j$
    else
        $Q_i(s,a_i) \leftarrow (1-\alpha)Q_i(s,a_i) + \alpha(c_j + min_a Q_i(s',a'))$
    end
    $d_i(s) \leftarrow min_a Q_i(s,a)$
    $\Delta \leftarrow max(\Delta, |\delta - d_i(s)|)$
    if $\Delta > \Theta$ then
        broadcast $\langle n_0, c_i, d_i(s)\rangle$
    end
end
**Tracking and Forwarding phase:**
received $\langle n_j, c_j, d_j\rangle$ at $i$ from node $j$ at link $a_i$
if $i == n_j$ then
    $n_i \leftarrow arg\ min_a Q_i(s,a)$
    broadcast $\langle n_i, c_i, d_i(s)\rangle$
else
    if $node\ i\ is\ parent\ of\ node\ j$ then
        $\delta \leftarrow min_a Q_i(s,a)$
        $Q_i(s,a_i) \leftarrow (1-\alpha)Q_i(s,a_i) + \alpha(c_j + min_a Q_i(s',a'))$
        $d_i(s) \leftarrow min_a Q_i(s,a)$
        $n_i \leftarrow argmin_a Q_i(s,a)$
        $\Delta \leftarrow max(\Delta, |\delta - d_i(s)|)$
        if $\Delta > \Theta$ then
            broadcast $\langle n_i, c_i, d_i(s)\rangle$
        end
    end
end
```

a packet is received from this link, $Q(s, a)$ is updated with the sum of the cost and distance metrics. Otherwise $Q(s, a)$ is updated according to Equation 4.6.

After updating, each node i evaluates its distance metric d_i using Equation 4.4. If the change in value of d_i is of a magnitude Δ larger than a fixed threshold θ, node i broadcasts a message to update its neighbors with its new c_i and d_i values. The condition $(\Delta \leq \Theta)$ serves as the stopping criterion for update of Q-values. Updating of Q-values continues in the initialization phase until all nodes do not notice any further changes in their Q-values.

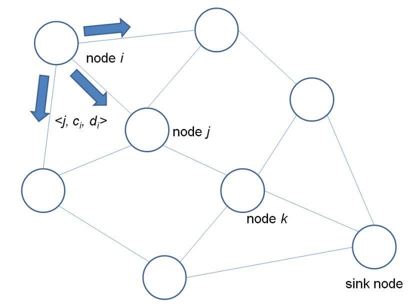
Data Forwarding Phase

In the tracking and forwarding phase, the source node is given by the distributed sensor election procedure described in section 4.4, which follows the state estimate of the detected target. After the source node has updated its posterior state estimate and elected its next sensor node, it transmits the updated state estimate towards the sink node via its *best* next-hop neighboring node. An ϵ-greedy policy is used, which is described in section 4.6.1.

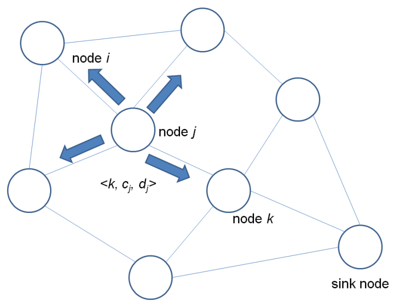
The message exchange between nodes for multi-hop routing is illustrated in Figure 4.4, for which node i is taken to be the current node. The procedure is as follows:

1. Node i forwards to node j which has the least Q-value (least expected sum of costs to the sink node), based on an ϵ-greedy policy
2. Subsequently, node j forwards to node k, its best next-hop node in the same manner
3. Node i overhears node j's packet to node k so it updates its previous action with the cost metric c_j of node j for the corresponding Q-value entry $Q(s_i, a_i)$
4. Subsequently, node i evaluates its distance metric d_i. If there is a significant change Δ that exceeds a threshold Θ, node i broadcasts its cost and distance metrics for neighboring nodes to update their respective Q-values.

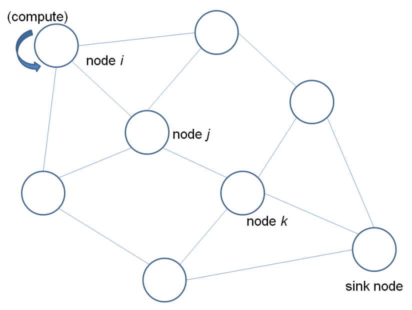
In the ϵ-greedy policy, the probability of transmitting to a random neighboring node, ϵ, is decreased with each tracking episode so that nodes increasingly exploit their learned policies to perform energy-efficient multihop routing.



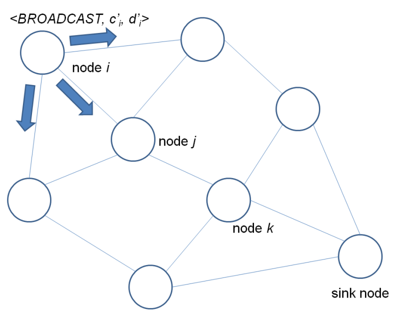
(a) Node i forwards to node j



(b) Node j forwards to node k



(c) Node i updates with cost c_j



(d) Node i broadcasts updated c_i', d_i'

Figure 4.4: Forwarding mechanism

4.7 Simulation Study

4.7.1 Simulation Setup

We simulated 100 nodes in a 10x10 sensor grid configuration, with node locations perturbed from grid points with a uniform distribution. The initial energy levels of nodes were uniformly distributed between 1450 and 1500 energy units. The target moved in a circular trajectory with a radius of four grid units, and target location estimates were computed using an Extended Kalman Filter (EKF) algorithm. The distributed sensor election procedure described in Section 4.4.1 was used to find the most suitable source node at each timestep, as well as its back-off delay and dynamic sensing interval. After the source node updated the state estimate with its measurement, it sent the state estimate to the sink node via multi-hop routing.

Each completed round of the target trajectory was considered as an episode for reinforcement learning, and simulations were run for 500 episodes. The episodic nature was required in order for the reinforcement learning algorithm to learn to make better forwarding decisions to neighbouring nodes, and the energy levels of nodes were assumed to be reset before the start of each episode. In real life, the resetting of nodes' energy levels could be justified from energy-harvesting mechanisms which allowed nodes' batteries to be re-charged before every episode. In our reinforcement learning mechanism, the probability of random action selection was given an initial value of $\epsilon = 0.9$, which was decreased with each episode.

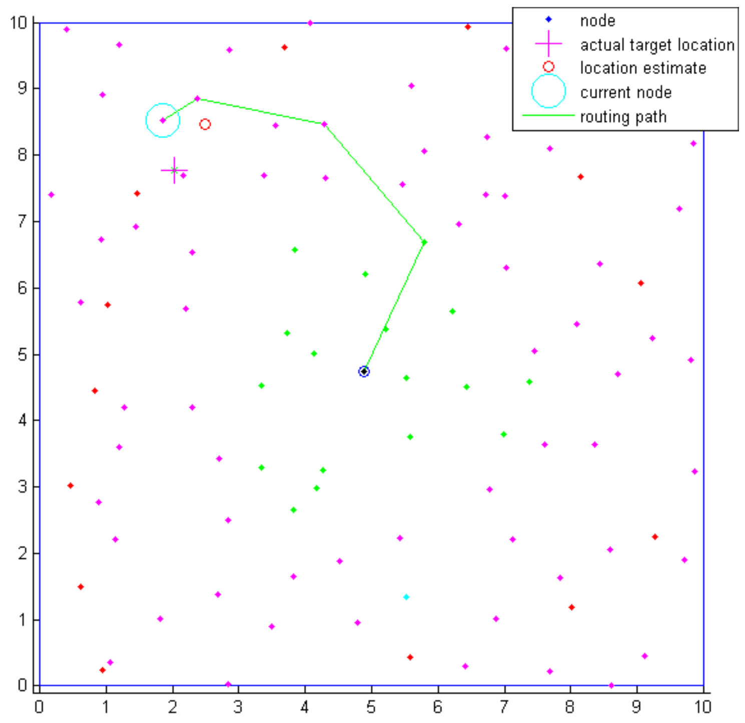


Figure 4.5: Multi-Hop Routing

Figure 4.5 shows the scatter plot of node locations with the sink node in the middle. The node with a large circular outline indicates the current source node which performs sensing and location state estimation. The + sign represents the actual target location while the small circle represents the location estimate. A sequence of line segments represents the multi-hop routing path from source to sink node.

Simulations were run to compare the average values of sensor network lifetime, tracking error, and delivery ratio from the source to sink node. If a state estimate was not delivered to the sink node within five hops, it was considered to be lost, and all the nodes on that routing path were penalised. The lifetime was defined to be the least remaining energy level of the nodes. A small learning rate $\alpha = 0.1$ was used, and simulations were conducted for distributed sensor election with and without the delayed sensing mechanism presented in section 4.4.2.

In addition, simulations were conducted for different values of the parameter β which represents the trade-off between information gain and remaining energy for distributed sensor election. We used β values of $0, 0.5, 1.0$ and we re-state Equations 4.1 and 4.2 here for convenience.

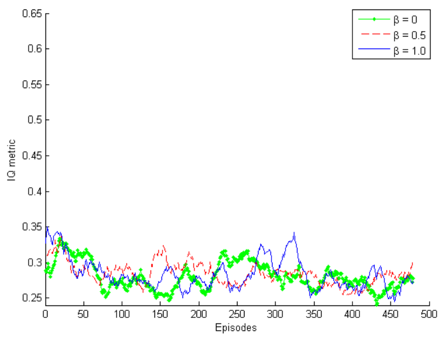
$$cost_i = \beta \frac{trace(\widehat{P}_{i,k+1|k+1})}{trace(\widehat{P}_{k+1|k})} - (1-\beta)\frac{e_i}{e_{max}} \tag{4.7}$$

$$\Delta t_{k,i} = \frac{(1 - cost_i) * \Delta T}{max(cost_i) - min(cost_i)} \tag{4.8}$$

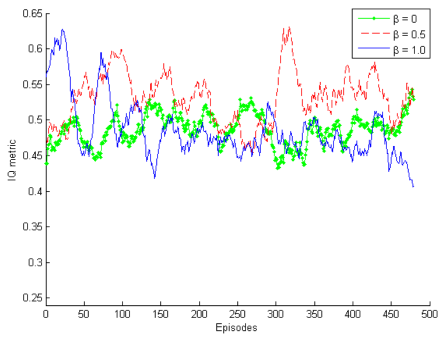
4.7.2 Results and Analysis

Comparing Covariance Traces

Figure 4.6 depicts the average amount of uncertainty in the updated posterior state estimates, based on the sensor selected in each timestep using Equation 4.7. Each data point in Figure 4.6 corresponds to the average covariance trace over all timesteps within one cycle of the target trajectory. As the covariance trace is a measure of uncertainty, lower values are more desirable. The covariance trace is independent of the multi-hop routing process, and it only affects Equation 4.7 in computing the composite cost values of each candidate node, so as to determine its back-off interval for distributed sensor election in 4.8.



(a) Results without delayed sensing



(b) Results with delayed sensing

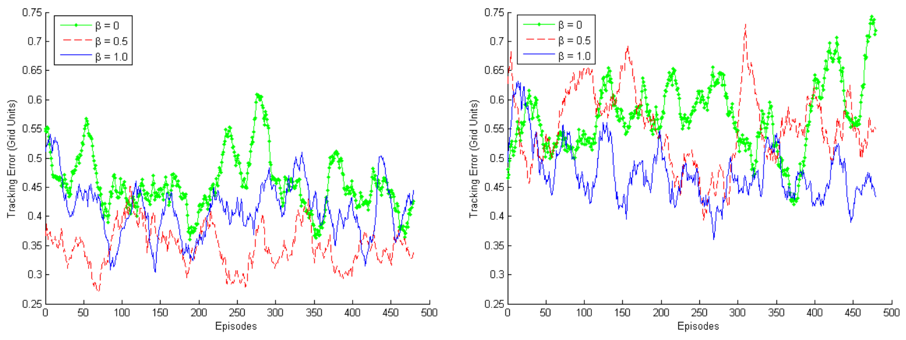
Figure 4.6: Comparison of average trace of covariance matrix

From Figure 4.6(a), no clear conclusions can be drawn from the average covariance traces for all three values of the parameter β. However, the graphs in Figure 4.6(b) show significantly higher covariance traces compared to those in Figure 4.6(a), due to the delayed sensing mechanism from section 4.4.2, which causes the winner node in the distributed sensor election procedure to incur additional sensing delay, so as to further conserve resources at the expense of increasing the estimation uncertainty.

The graph for $\beta = 1.0$ lies below that for $\beta = 0$ for most parts of Figure 4.6(b) as the high emphasis on IQ selects sensors which are better at reducing estimation uncertainty. In addition, the graph for $\beta = 0.5$ is substantially higher than the other two. As $\beta = 0.5$ places equal emphasis on IQ and energy cost in Equation 4.7, this could result in selecting nodes with high remaining energy but high IQ cost in terms of estimation uncertainty.

Comparing Average Tracking Error

The impact that the trade-off parameter β, in the composite cost function (Equation 4.7), has on the selection of nodes is evident in the plots of tracking error in Figure 4.7. As $\beta = 1.0$ places emphasis entirely on tracking error, com-



(a) Results without delayed sensing (b) Results with delayed sensing

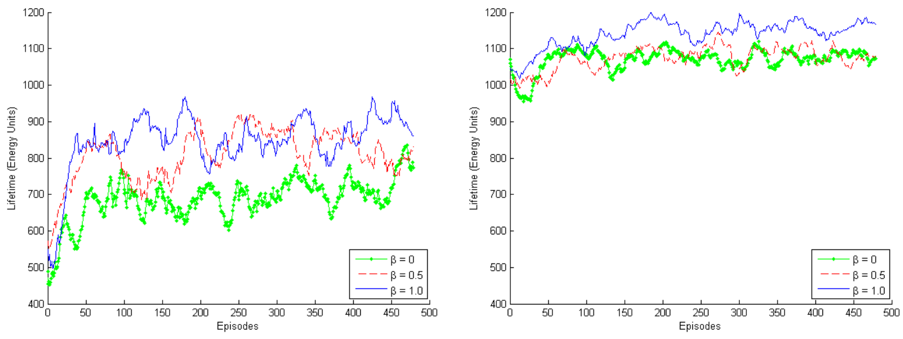
Figure 4.7: Comparison of average tracking error in grid units

pared to $\beta = 0$ which emphasizes the remaining energy of candidate nodes, it is expected that the graph for $\beta = 1.0$ in Figure 4.7(a) consistently lies below the graph for $\beta = 0$. In addition, the graph for $\beta = 0.5$, which places equal emphasis on IQ and energy costs, lies substantially lower than the other two graphs in Figure 4.7(a). One possible explanation could be that nodes with low IQ costs also happen to have high levels of remaining energy.

The effect of the parameter β in Figure 4.7(b) is even more interesting, as the delayed sensing mechanism in section 4.4.2 allows elected sensor nodes to delay sensing in order to conserve energy, thus increasing the remaining energy levels in nodes. Since, $\beta = 0.5$ places equal emphasis on IQ and energy costs, the higher levels of remaining energy skews Equation 4.7 in favor of nodes with high remaining energy. This could result in selecting nodes with low IQ costs, as previously observed in Figure 4.6(b), in which the average covariance traces of selected nodes were significantly higher.

As a result of choosing nodes with high remaining energy but little information gain, the tracking error for $\beta = 0.5$ in Figure 4.7(b) is significantly increased, so much so that at some points it is no different from using $\beta = 0$. Similar to Figure 4.7(a), the graph for $\beta = 1.0$ lies consistently below the graph for $\beta = 0$ in Figure 4.7(b). Comparing Figures 4.7(a) and 4.7(b) validates that the delayed sensing mechanism conserves the remaining energy levels of sensor nodes at the expense of increased tracking error.

Comparing Average Lifetime



(a) Results without delayed sensing

(b) Results with delayed sensing

Figure 4.8: Comparison of average sensor network lifetime in energy units

Figure 4.8 shows our simulation results for average sensor network lifetime, which we define as the time until the first node dies. We use the minimum remaining energy level of nodes to estimate the remaining network lifetime. The higher network lifetimes for all three graphs in Figure 4.8(b) over the respective graphs in Figure 4.8(a) can be attributed to the sensing delay mechanism, which conserves the remaining energy levels of the elected node and its neighboring nodes at each timestep.

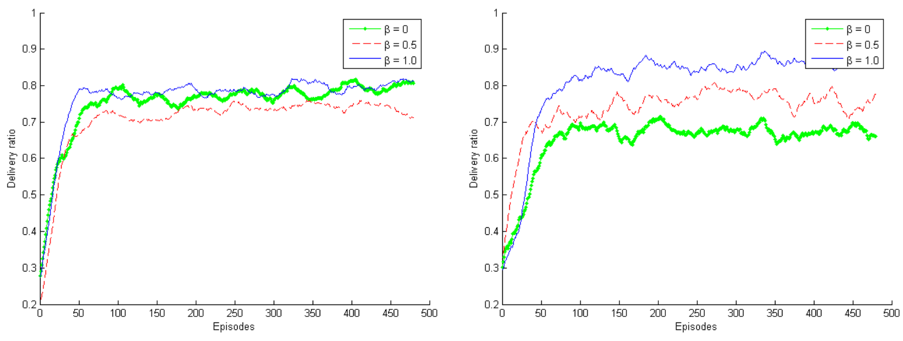
As β increases from 0 to 1.0 increasing emphasis is placed on IQ cost in Equation 4.7. However, the graphs in Figures 4.8(a) and (b) show that increasing the value of β actually increases the remaining lifetime. This is attributed to Equation 4.8, which computes the back-off interval for distributed sensor election. As increasing emphasis is placed on reducing IQ cost as β is increased, nodes with high IQ cost elect themselves with a higher back-off delay, thus effectively increasing the sensing interval and *reducing* the energy consumption.

As a result, this explains why for both Figures 4.8(a) and (b), the graphs for $\beta = 1.0$ consistently show higher remaining lifetime as compared to the graphs for $\beta = 0$. Our mechanism of sensor election back-off in Equation 4.8 accounts for this interesting observation, which may seem counter-intuitive at first glance.

In addition, the graph for $\beta = 0.5$ is closer to that for $\beta = 1.0$ in Figure 4.8(a), but closer the graph for $\beta = 0$ in Figure 4.8(b). This indicates that when the delayed sensing mechanism is invoked to help to conserve energy, the effect of $\beta = 0.5$ is quite similar to that of $\beta = 0$, in that remaining energy levels of nodes play a higher weightage in Equation 4.7, so much so that nodes' IQ costs play a little role in sensor election. In contrast, when the remaining energy levels are lower in Figure 4.8(a), equal weightage of IQ and energy costs brought about by $\beta = 0.5$ actually skews Equation 4.7 closer to nodes' IQ cost.

Comparing Delivery Ratio

Figure 4.9 shows the graphs for delivery ratio, which is given by the number of state estimate update packets generated at the source node, that are delivered to the sink node for each round of the target trajectory. Packets which are not delivered to the sink node by five hops are discarded, as the these state estimates have become outdated. Nodes that lie along these routing paths are penalised with significant cost, so that they can adjust their Q-values and gradually learn to make better and better decisions in forwarding to their neighboring nodes.



(a) Results without delayed sensing (b) Results with delayed sensing

Figure 4.9: Comparison of delivery rate to sink node

The effect of the reinforcement learning process is evident in both sets of graphs in Figure 4.9, in which the delivery ratios start from very low values, during which nodes randomly forward packets in an exploration phase to learn their Q-values, and quickly increase in the first fifty episodes, as nodes start to exploit their learnt Q-values to make better forwarding decisions. For both sets of graphs, the learning process begins to saturate at around the 100th episode.

From Figures 4.8 and 4.9, there is a relation between remaining lifetime and delivery ratio, in that the graphs for $\beta = 1.0$ place total emphasis on IQ cost, resulting in higher remaining lifetime and delivery ratio, as compared to the graphs for $\beta = 0$, which emphasize the remaining energy of nodes. As explained in the previous section, a higher β value increases the back-off delay in Equation 4.8, resulting in a greater number of nodes with high remaining energy being available for forwarding packets to the sink node.

If nodes around the sink node have low energy, sensors tend to make locally-optimal decisions to forward to nodes with higher energy, which may result in routing *away* from the sink node. Thus the remaining lifetime plays a significant part in multi-hop routing to the sink node. This is further illustrated by comparing the graphs for $\beta = 1.0$ between Figures 4.9(a) and (b), in which Figure 4.9(b) has a higher delivery ratio due to the delayed sensing mechanism, which helps to conserve the remaining energy of nodes.

The graphs for $\beta = 0.5$ in both Figures 4.9(a) and (b) are comparable, indicating a small effect in the delayed sensing mechanism on the delivery ratio. On the other hand, the graph for $\beta = 0$ in Figure 4.9(b) is significantly lower than that for Figure 4.9(a), as the distributed sensor election procedure emphasizing the remaining energy of nodes actually results in a significantly smaller number of nodes with sufficient remaining energy to route towards the sink node, thus reducing the remaining lifetime.

4.8 Discussions

In this chapter, we have addressed distributed sensor election and multi-hop routing for target tracking in wireless sensor networks. First, we adopted a distributed sensor election procedure that considers the trade-off between IQ (information-quality) and energy costs of candidate nodes, in order to set different back-off values for the sensing intervals. Nodes that win the distributed sensor election procedure with larger back-off delays tend to conserve energy better.

Subsequently, we have also addressed energy-aware multi-hop routing to the sink node for which a cost metric based on nodes' remaining energy levels was used to convert energy-aware routing to a minimum-cost problem. In our approach, the benefit of applying reinforcement learning to perform energy-aware multi-hop routing to the sink node is that nodes gradually learn which of their neighboring nodes to forward in a distributed manner, based on their remaining energy levels and local one-hop information. Nodes are not aware of where the sink node is, and each node only maintains a distance metric that is updated with cost feedback from neighboring nodes.

As each node learns to make better and better forwarding decisions, using a combination of *exploration* of new neighbors to find new solutions, and *exploitation* of the best solution known so far, global information in the form of the distance vector is distributed among nodes, which nodes update one another with local information. After some iterations, nodes making locally-optimal decisions to forward to their next-hop neighbours gradually converge to a global optimal solution of energy-aware multi-hop routing to the sink node.

Chapter 5

Conclusions

In this thesis, we have addressed different aspects of resource management for target tracking in wireless sensor networks.

First, we provided an overview of the Extended Kalman Filter as a tracking algorithm, and we presented its implementation in a real-world wireless sensor network test-bed with system design considerations. We compared the performance of different process models and sensor selection schemes for a single-hop sensor deployment, and we used the trace of the EKF covariance matrix as an *information quality* (IQ) metric for sensor selection. We extended our clustered system architecture design to include mobile devices such as smartphones, for real-time remote monitoring and visualization, which could be extended to various applications for indoor tracking.

In the second part of this thesis, we performed a simulation study of distributed IQ-based sensor election and multi-hop routing to a sink node. The sensor election approach was extended from [1] and we used a composite cost function to trade-off IQ with remaining energy of candidate nodes, with a weight parameter β. In addition, the winner node in the distributed sensor election procedure could introduce additional sensing delay based on its IQ metric, so as to conserve more energy, subject to IQ constraints.

Subsequently, we addressed the issue of energy-aware multi-hop routing from source to sink node by using a cost function to convert the remaining energy of nodes into a cost metric, for which an additive expected sum of costs was used to make forwarding decisions to perform minimum-cost routing. Reinforcement learning was applied to learn to forward packets to the sink node and increase the delivery ratio. Our simulations compared tracking error, network lifetime and delivery ratio for different values of the trade-off parameter β, as well as for distributed sensor election with and without the delayed sensing mechanism.

Although the IQ-based sensor election procedure was initially designed as a resource management approach separate from the energy-aware multi-hop routing mechanism, the composite cost function that was used to trade-off IQ and energy cost, and subsequently decide the back-off delay, had a significant effect on the performance of the multi-hop routing algorithm. It was observed that, based on different values of β, the combination of IQ and error costs that had the least composite cost, would result in a larger back-off delay, that could help to conserve remaining energy levels of nodes. In that manner, even though emphasis was placed on maximizing IQ, a significantly large back-off delay also resulted in increasing network lifetime. As a result of larger remaining lifetime, the delivery ratio to the sink node was increased, as there were more nodes with higher remaining energy to route to the sink node.

Such coupling of seemingly unrelated performance metrics as a result of our composite cost function and back-off mechanism provides many interesting new avenues for multi-objective decision-making in our future work. In addition, we could look at using a distributed value function approach [36] to speed up reinforcement learning, such that neighbors can update their Q-values from overhearing cost feedback messages, even if they have not taken any actions. In addition, other forms of the cost function in our reinforcement learning system model can be studied in future work.

Bibliography

[1] Y. K. Toh, W. Xiao, and L. Xie, “A Wireless Sensor Network Target Tracking System with Distributed Competition based Sensor Scheduling,” in *Proceedings of the 2007 International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP*, pp. 257–262, 2007.

[2] V. T. Pham, Q. Qiu, A. A. P. Wai, and J. Biswas, “Application of Ultrasonic Sensors in a Smart Environment,” *Journal of Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 180–207, 2007.

[3] A. Roy, S. K. Das, and K. Basu, “A Predictive Framework for Location-Aware Resource Management in Smart Homes,” *IEEE Transactions on Mobile Computing*, vol. 6, pp. 1270–1283, 2007.

[4] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, “Next Century Challenges: Scalable Coordination in Sensor Networks,” in *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 263–270, 1999.

[5] F. Zhao, J. Shin, and J. Reich, “Information-Driven Dynamic Sensor Collaboration for Tracking Applications,” *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 61–72, 2002.

[6] J. L. Williams, J. W. Fisher, and A. S. Willsky, “Approximate Dynamic Programming for Communication-Constrained Sensor Network Manage-

ment," *IEEE Transactions on Signal Processing*, vol. 55, no. 8, pp. 4300–4311, 2007.

[7] C. M. Kreucher, D. Blatt, A. O. Hero, and K. Kastella, "Adaptive Multimodality Sensor Scheduling for Detection and Tracking of Smart Targets," *Digital Signal Processing*, vol. 16, no. 5, pp. 546–567, 2006.

[8] L.-L. S. Ong, *Non-Gaussian Representations for Decentralised Bayesian Estimation.* PhD thesis, School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, 2007.

[9] R. Brooks, P. Ramanathan, and A. Sayeed, "Distributed Target Classification and Tracking in Sensor Networks," *Proceedings of the IEEE*, vol. 91, pp. 1163–1171, Aug 2003.

[10] Y. Yu and V. K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," *ACM Springer Mobile Networks and Applications (MONET) Journal*, vol. 10, no. 1-2, pp. 115–131, 2005.

[11] H. Park and M. B. Srivastava, "Energy-Efficient Task Assignment Framework for Wireless Sensor Networks," *CENS Technical Report*, September 2003.

[12] Y. Tian, E. Ekici, and F. Özgüner, "Cluster-based information processing in wireless sensor networks: an energy-aware approach," *Journal of Wireless Communications & Mobile Computing*, vol. 7, no. 7, pp. 893–907, 2007.

[13] K. Shah and M. Kumar, "Distributed Independent Reinforcement Learning (DIRL) Approach to Resource Management in Wireless Sensor Networks," in *IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2007. MASS 2007*, pp. 1–9, Oct 2007.

[14] J. Polastre, J. Hill, and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems (2004), SenSys'04*, pp. 95–107, Nov 2004.

[15] W. Ye, J. Heidemann, and D. Estrin, "An Energy-efficient MAC Protocol for Wireless Sensor Networks," in *21st Conference of the IEEE Computer and Communications Societies (INFOCOM)*, vol. 3, pp. 1567–1576, Jun 2002.

[16] Y. Tay, K. Jamieson, and H. Balakrishnan, "Collision-Minimizing CSMA and its Applications to Wireless Sensor Networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 1048–1057, Aug 2004.

[17] K. Klues, G. Hackmann, O. Chipara, and C. Lu, "A Component-Based Architecture for Power-Efficient Media Access Control in Wireless Sensor Networks," in *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pp. 59–72, 2007.

[18] J. N. Al-Karaki and A. E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 6–28, 2004.

[19] M. Chu, H. Haussecker, and F. Zhao, "Scalable Information-Driven Sensor Querying and Routing for Ad Hoc Heterogeneous Sensor Networks," *International Journal of High Performance Computing Applications*, vol. 16, 2002.

[20] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 56–67, 2000.

[21] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive Protocols for Information Dissemination in Wireless Sensor Networks," in *MobiCom*

'99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, pp. 174–185, 1999.

[22] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," *Technical Report: TR95-041, University of North Carolina at Chapel Hill*, 2001.

[23] W. Xiao, J. Wu, L. Xie, and L. Dong, "Sensor Scheduling for Target Tracking in Networks of Active Sensors," in *ACTA AUTOMATICA SINICA*, vol. 32, pp. 173–180, 2006.

[24] W.-L. Yeow, C.-K. Tham, and W.-C. Wong, "Energy Efficient Multiple Target Tracking in Wireless Sensor Networks," *IEEE Transactions on Vehicular Technology*, vol. 56, no. 2, pp. 918–928, 2007.

[25] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking," *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188, 2001.

[26] E. F. Nakamura, A. A. F. Loureiro, and A. C. Frery, "Information Fusion for Wireless Sensor Networks: Methods, Models, and Classifications," *ACM Computing Surveys*, vol. 39, no. 3, 2007.

[27] N. Xiong and P. Svensson, "Multi Sensor Management for Information Fusion: Issues and Approaches," *Information Fusion*, vol. 3, no. 2, pp. 163–186, 2002.

[28] K. Akkaya and M. Younis, "A Survey on Routing Protocols for Wireless Sensor Networks," *Ad Hoc Networks*, vol. 3, pp. 325–349, 2005.

[29] J. Kulik, W. Heinzelman, and H. Balakrishnan, "Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks," *Journal of Wireless Networks*, vol. 8, no. 2/3, pp. 169–185, 2002.

[30] J.-H. Chang and L. Tassiulas, "Maximum Lifetime Routing In Wireless Sensor Networks," *IEEE/ACM Transactions on Networking*, vol. 12, pp. 609–619, Aug 2004.

[31] F. Ye, A. Chen, S. Lu, and L. Zhang, “A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks,” in *Proceedings of the Tenth International Conference on Computer Communications and Networks, 2001*, pp. 304–309, 2001.

[32] W. Naruephiphat and W. Usaha, “Balancing Tradeoffs for Energy-Efficient Routing in MANETs Based on Reinforcement Learning,” in *Proceedings of IEEE Vehicular Technology Conference, 2008. VTC Spring 2008*, pp. 2361–2365, May 2008.

[33] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich, “Collaborative Signal and Information Processing: An Information Directed Approach,” *Proceedings of the IEEE*, vol. 91, pp. 1199–1209, Aug 2003.

[34] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 1994.

[35] D. P. Bertsekas, *Dynamic Programming and Optimal Control.* Athena Scientific, 3rd ed., 2007.

[36] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[37] E. Altman, *Constrained Markov Decision Processes.* Chapman and Hall, 1999.

[38] Y. He and E. Chong, “Sensor Scheduling for Target Tracking in Sensor Networks,” in *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC’04)*, pp. 743–748, 2004.

[39] Y. Li, L. W. Krakow, E. K. P. Chong, and K. N. Groom, “Approximate stochastic dynamic programming for sensor scheduling to track multiple targets,” in *Proceedings of the 2006 Workshop on Defense Applications of Signal Processing (DASP’06)*, 2006.

[40] E. K. P. Chong, C. Kreucher, and A. O. H. III, "Monte-Carlo-based Partially Observable Markov Decision Process Approximations for Adaptive Sensing," in *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES'08)*, pp. 173–180, 2008.

[41] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models.* Simon and Schuster, first ed., 1978.

[42] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[43] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming (Optimization and Neural Computation Series, 3).* Athena Scientific, 1996.

[44] N. Lilith, K. Dogancay, and G. Ibal, "Dynamic Sensor Scan Optimisation using Reinforcement Learning," in *Proceedings of the 2007 International Conference on Intelligent Sensors, Sensor Networks and Information Processing, ISSNIP*, pp. 407–412, 2007.

[45] C.-K. Tham, *Modular On-line Function Approximation for Scaling Up Reinforcement Learning.* PhD thesis, University of Cambridge, 1994.

[46] D. A. Castañon, "Approximate Dynamic Programming for Sensor Management," in *Proceedings of the 36th IEEE Conference on Decision and Control, 1997*, pp. 1202–1207, 1997.

[47] S. Aeron, V. Saligrama, and D. A. Castañon, "Efficient Sensor Management Policies for Distributed Target Tracking in Multihop Sensor Networks," *IEEE Transactions on Signal Processing*, vol. 56, no. 6, pp. 2562–2574, 2008.

[48] S. Reddy, J. Burke, D. Estrin, M. H. Hansen, and M. B. Srivastava, "A Framework for Data Quality and Feedback in Participatory Sensing," in *Proceedings of the 5th International Conference on Embedded Networked*

Sensor Systems, SenSys 2007, Sydney, NSW, Australia, November 6-9, 2007, pp. 417–418, 2007.

[49] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt, “Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation,” in *MobiSys ’08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pp. 174–186, 2008.

[50] `http://www.xbow.com/Products/wproductsoverview.aspx`.

[51] `http://www.easysen.com/`.

[52] `http://www.tinyos.net/`.

[53] L. D. Stone, T. L. Corwin, and C. A. Barlow, *Bayesian Multiple Target Tracking.* Artech House, Inc., 1999.

[54] T. Choudhury, G. Borriello, S. Consolvo, D. Haehnel, B. Harrison, B. Hemingway, J. Hightower, P. P. Klasnja, K. Koscher, A. LaMarca, J. A. Landay, L. LeGrand, J. Lester, A. Rahimi, A. Rea, and D. Wyatt, “The Mobile Sensing Platform: An Embedded Activity Recognition System,” *IEEE Pervasive Computing*, vol. 7, pp. 32–41, 2008.

[55] S. Consolvo, D. W. McDonald, T. Toscos, M. Y. Chen, J. Froehlich, B. L. Harrison, P. V. Klasnja, A. LaMarca, L. LeGrand, R. Libby, I. E. Smith, and J. A. Landay, “Activity Sensing in the Wild: A Field Trial of Ubifit Garden,” in *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pp. 1797–1806, 2008.

[56] `http://developer.android.com/guide/basics/what-is-android.html`.

[57] `http://cnds.ece.nus.edu.sg/uwb-sc/`.

[58] S. Haykin, *Neural Networks and Learning Machines.* Pearson, third ed., 2009.

[59] J. Liu, F. Zhao, and D. Petrovic, “Information-Directed Routing in Ad Hoc Sensor Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 23, pp. 851–861, Apr 2005.

[60] J. A. Boyan and M. L. Littman, “Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach,” in *Advances in Neural Information Processing Systems 6*, pp. 671–678, 1993.

[61] Y. Zhang, M. P. J. Fromherz, and L. D. Kuhn, “Smart Routing with Learning-based QoS-aware Routing Strategies,” in *First Workshop on QoS Routing*, pp. 298–307, Oct 2004.

[62] Y. Zhang and Q. Huang, “A Learning-based Adaptive Routing Tree for Wireless Sensor Networks,” *Journal of Communications*, vol. 1, no. 2, pp. 12–21, 2006.

[63] Y. Zhang, J. Liu, and F. Zhao, “Information-Directed Routing in Sensor Networks Using Real-Time Reinforcement Learning,” *Combinatorial Optimization in Communication Networks, Springer*, 2006.

[64] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-down Approach featuring the Internet.* Pearson Addison-Wesley, 2004.