FULLY ADAPTIVE RADAR MODELING AND SIMULATION DEVELOPMENT

Kristine L. Bell and Anthony Kellems
Metron, Inc.

Graeme E. Smith
The Ohio State University

Bruce L. McKinley
Signal Processing Consultants, Inc.

APRIL 2017
Final Report

THIS IS A SMALL BUSINESS INNOVATION RESEARCH (SBIR) PHASE I REPORT.

Approved for public release; distribution unlimited.

See additional restrictions described on inside pages

STINFO COPY

AIR FORCE RESEARCH LABORATORY
SENSORS DIRECTORATE
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7320
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE
NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the USAF 88th Air Base Wing (88 ABW) Public Affairs Office (PAO) and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RY-WP-TR-2017-0074 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

// Signature//            // Signature//
Muralidhar Rangaswamy  Gregory Cazzell,  Chief
Program Manager  Radio Frequency Exploitation Branch
Radio Frequency Exploitation Branch

// Signature//
Doug Hager, Deputy
Layered Sensing Exploitation Division

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government’s approval or disapproval of its ideas or findings.

*Disseminated copies will show “//Signature//” stamped or typed above the signature blocks.*
NOTICE TO ACCOMPANY THE DISSEMINATION OF EXPORT CONTROLLED TECHNICAL DATA

Export of the attached information (which includes, in some circumstances, release to foreign nationals within the United States) without first obtaining approval or license from the Department of State for items controlled by the International Traffic in Arms Regulation (ITAR), or the Department of Commerce for items controlled by the Export Administration Regulation (EAR), may constitute a violation of law.

Under 22 U.S.C. 2778, the penalty for unlawful export of items or information controlled under the ITAR is up to ten years imprisonment, or a fine of $1,000,000, or both. Under 50 U.S.C., appendix 2410, the penalty for unlawful export of items or information controlled under the EAR is a fine of up to $1,000,000, or five times the value of the exports, whichever is greater; or for an individual, imprisonment of up to 10 years, or fine of up to $250,000, or both.

In accordance with your certification that establishes you as a “qualified U.S. contractor,” unauthorized dissemination of this information is prohibited and may result in your disqualification as a qualified U.S. contractor, and may be considered in determining your eligibility for future contracts with the Department of Defense.

The U.S. Government assumes no liability for direct patent infringement, or contributory patent infringement, or misuse of technical data.

The U.S. Government does not warrant the adequacy, accuracy, currency, or completeness of the technical data.

The U.S. Government assumes no liability for loss, damage, or injury resulting from manufacture or use for any purpose of any product, article, system, or material involving reliance upon any or all technical data furnished in response to the request for technical data.

If the technical data furnished by the Government will be used for commercial manufacturing or other profit potential, a license for such use may be necessary. Any payments in support of the request for data do not include or involve any license rights.

A copy of this notice shall be provided with any partial or complete reproduction of these data that are provided to qualified U.S. contractors.
# REPORT DOCUMENTATION PAGE

**1. REPORT DATE** (DD-MM-YY)  
April 2017

**2. REPORT TYPE**  
Final

**3. DATES COVERED** (From - To)  
9 June 2016 – 9 January 2017

**4. TITLE AND SUBTITLE**  
FULLY ADAPTIVE RADAR MODELING AND SIMULATION DEVELOPMENT

**5a. CONTRACT NUMBER**  
FA8650-16-M-1774

**5b. GRANT NUMBER**  
N/A

**5c. PROGRAM ELEMENT NUMBER**  
65502F

**5d. PROJECT NUMBER**  
3005

**5e. TASK NUMBER**  
N/A

**5f. WORK UNIT NUMBER**  
Y1HV

**6. AUTHOR(S)**  
Kristine L. Bell and Anthony Kellems (Metron, Inc.)  
Graeme E. Smith (The Ohio State University)  
Bruce L. McKinley (Signal Processing Consultants, Inc.)

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**  
Metron, Inc.  
1818 Library St., Suite 600  
Reston, VA 20190

The Ohio State University  
Signal Processing Consultants, Inc.

**8. PERFORMING ORGANIZATION REPORT NUMBER**  
N/A

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**  
Air Force Research Laboratory  
Sensors Directorate  
Wright-Patterson Air Force Base, OH 45433-7320  
Air Force Materiel Command  
United States Air Force

**10. SPONSORING/MONITORING AGENCY ACRONYM(S)**  
AFRL/R

**11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)**  
AFRL-RY-WP-TR-2017-0074

**12. DISTRIBUTION/AVAILABILITY STATEMENT**  
Approved for public release; distribution is unlimited.

**13. SUPPLEMENTARY NOTES**  

**14. ABSTRACT**  
This is a Small Business Innovation Research (SBIR) Phase I Report developed under SBIR contract for topic AF161-132 Fully Adaptive Radar Modeling and Simulation Development. We have developed a MATLAB-based modeling and simulation (M&S) architecture for distributed fully adaptive radar (FAR) that will enable algorithm development and testing on simulated, previously collected, and real-time streaming data. The architecture is coded in MATLAB using an object oriented programming approach. The architecture includes a FAR engine to control the operation of the perception-action cycle and software objects that determine the next set of sensing parameters; obtain data from the sensor; process the data to track the target; and store and display the results of the sensing and tracking processes. We have developed modules that implement simulated and pre-recorded software defined radar (SDR) data examples, and real-time and simulated data examples from the Cognitive Radar Engineering Workspace (CREW) at The Ohio State University. The FAR M&S architecture allows for transparent switching between the simulated and experimental CREW data sources, as well as between FAR algorithms that drive the sensing. The ability to easily interchange sensing and processing modules will allow for rapid development and testing of cognitive radar algorithms by structuring the M&S functions to avoid duplicating effort and “single point” solutions. It will enable collaboration between researchers in industry, academia, and the Air Force, as algorithms developed by different researchers can be tested and compared using consistent simulations, collected data, and laboratory conditions. Report contains color.

**15. SUBJECT TERMS**  
SBIR report, fully adaptive radar, cognitive radar, modeling and simulation

**16. SECURITY CLASSIFICATION OF:**  
a. REPORT  
Unclassified  
b. ABSTRACT  
Unclassified  
c. THIS PAGE  
Unclassified

**17. LIMITATION OF ABSTRACT:**  
SAR

**18. NUMBER OF PAGES**  
50

**19a. NAME OF RESPONSIBLE PERSON (Monitor)**  
Muralidhar Rangaswamy

**19b. TELEPHONE NUMBER (Include Area Code)**  
N/A

*Standard Form 298 (Rev. 8-98)*  
Prescribed by ANSI Std. Z39-18
9 February 2017

Dr. Muralidhar Rangaswamy  
AFRL/RYAP  
2241 Avionics Circle  
Wright-Patterson AFB, OH 45433  

Subject: Contract Number FA8650-16-M-1774, Phase I SBIR Topic AF161-132  
Fully Adaptive Radar Modeling and Simulation Development  
Report # TBD  

Dear Dr. Rangaswamy:

Metron, Inc. hereby waives its SBIR Data Rights to all contents of the final report for subject contract. The Government is granted an unlimited nonexclusive license to use, modify, reproduce, release, perform, and display or disclose this report and the data contained herein.

Sincerely,

Kristine L. Bell, Ph.D.  
Senior Scientist  
Advanced Mathematics Applications Division  
Metron, Inc.  
bell@metsci.com  
703-326-2913
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>iii</td>
</tr>
<tr>
<td>1 SUMMARY</td>
<td>1</td>
</tr>
<tr>
<td>2 INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>3 METHODS, ASSUMPTIONS, AND PROCEDURES</td>
<td>5</td>
</tr>
<tr>
<td>3.1 Fully Adaptive Radar Framework</td>
<td>5</td>
</tr>
<tr>
<td>3.1.1 General Framework</td>
<td>5</td>
</tr>
<tr>
<td>3.1.2 Cognitive Single Target Tracking</td>
<td>8</td>
</tr>
<tr>
<td>3.1.3 Cognitive Single Target Tracking, Initiation, and Termination</td>
<td>9</td>
</tr>
<tr>
<td>3.1.4 Cognitive MAP-PF Single Target Tracking</td>
<td>14</td>
</tr>
<tr>
<td>3.2 Examples</td>
<td>18</td>
</tr>
<tr>
<td>3.3 Summary</td>
<td>18</td>
</tr>
<tr>
<td>4 RESULTS AND DISCUSSION</td>
<td>19</td>
</tr>
<tr>
<td>4.1 FAR M&amp;S Software Architecture Overview</td>
<td>19</td>
</tr>
<tr>
<td>4.2 Using the FAR M&amp;S MATLAB Code</td>
<td>21</td>
</tr>
<tr>
<td>4.2.1 System Requirements</td>
<td>21</td>
</tr>
<tr>
<td>4.2.2 Getting Started</td>
<td>21</td>
</tr>
<tr>
<td>4.3 FAR M&amp;S Architecture Base Classes</td>
<td>22</td>
</tr>
<tr>
<td>4.3.1 Scene</td>
<td>22</td>
</tr>
<tr>
<td>4.3.2 Sensor</td>
<td>22</td>
</tr>
<tr>
<td>4.3.3 Processor</td>
<td>22</td>
</tr>
<tr>
<td>4.3.4 Optimizer</td>
<td>22</td>
</tr>
<tr>
<td>4.3.5 Storage Manager</td>
<td>22</td>
</tr>
<tr>
<td>4.3.6 Display Manager</td>
<td>23</td>
</tr>
<tr>
<td>4.3.7 Timing Manager</td>
<td>23</td>
</tr>
<tr>
<td>4.3.8 Perception Action Cycle</td>
<td>23</td>
</tr>
<tr>
<td>4.3.9 FAR Engine</td>
<td>27</td>
</tr>
<tr>
<td>4.3.10 Utility Functions</td>
<td>27</td>
</tr>
<tr>
<td>4.4 The CREW Sensor</td>
<td>28</td>
</tr>
<tr>
<td>4.5 Summary</td>
<td>31</td>
</tr>
<tr>
<td>5 CONCLUSIONS</td>
<td>32</td>
</tr>
<tr>
<td>6 REFERENCES</td>
<td>33</td>
</tr>
<tr>
<td>LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS</td>
<td>40</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: Cognitive Sensor/Processor System Framework</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2: Cognitive MAP-PF Sensor/Processor System Framework</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3: FAR M&amp;S Architecture Block Diagram</td>
<td>20</td>
</tr>
<tr>
<td>Figure 4: CREW Schematic Showing the Digital Backend, IF Stage, and W-band RF Stage</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5: CREW Sensor Object</td>
<td>30</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1: Cognitive Single Target Tracking Recursion</td>
<td>9</td>
</tr>
<tr>
<td>Table 2: Cognitive Single Target Tracking, Initiation, and Termination Recursion</td>
<td>13</td>
</tr>
<tr>
<td>Table 3: Cognitive Single Target MAP-PF Tracking Recursion</td>
<td>17</td>
</tr>
</tbody>
</table>
1 SUMMARY

The potential of cognitive approaches to enhance existing radar performance in almost all respects has led to an upsurge in research in recent years and a key gap in the Air Force’s radar modeling and simulation (M&S) tools is the lack of a comprehensive, dynamic distributed radar scenario generation capability for distributed fully adaptive radar (FAR) systems. As of early 2015, the research had all been advancing concepts theoretically and examining their performance through simulation, or at best using pre-recorded data. There had been no reports of experimentally validated concepts, largely because the necessary hardware to test them had not been developed. However, this step is vital in order to establish the true performance potential of applying cognitive processing methods. To address this, the Cognitive Sensing Laboratory (CSL) at The Ohio State University (OSU) ElectroScience Laboratory (ESL), in conjunction with Metron, Inc., the Air Force Research Laboratory (AFRL), and the Air Force Office of Scientific Research (AFOSR), has embarked on a program of research to develop and examine cognitive radar processing concepts analytically and experimentally.

The CSL designed and built the Cognitive Radar Engineering Workspace (CREW), the world’s first radar testbed built specifically to allow testing of fully adaptive and cognitive algorithms, and Metron and OSU have developed a theoretical framework for a cognitive FAR system that identifies and mathematically models the key system components within the context of target detection and tracking for a single sensor and target. We have been developing modeling, simulation, analysis, and experimentation capabilities to demonstrate the performance improvement achieved by FAR systems over traditional feed-forward radar (FFR) systems. We began with simulated scenarios and pre-recorded data from OSU’s Software Defined Radar (SDR) system. We now have the capability to demonstrate the real-time operation of the cognitive radar tracking system using the CREW.

The goal of this project is to develop a MATLAB-based M&S architecture for distributed FAR radar that will enable algorithm development and testing on simulated, previously collected, and real-time streaming data. In Phase I, we have developed a baseline FAR M&S architecture that is coded in MATLAB using an object oriented programming (OOP) approach. It includes a FAR engine to control the operation of the perception-action (PA) cycle and software objects that determine the next set of sensing parameters; obtain data from the sensor; process the data to track the target; and store and display the results of the sensing and tracking processes. We have developed modules that implement simulated and pre-recorded SDR data examples, and real-time and simulated CREW data examples.

The FAR M&S architecture developed in Phase I allows for transparent switching between the simulated and experimental CREW data sources, as well as between FAR algorithms that drive the sensing. The ability to easily interchange sensing and processing objects will allow for rapid development and testing of cognitive radar algorithms by structuring the M&S functions to avoid duplicating effort and “single point” solutions. It will enable collaboration between researchers in industry, academia, and the Air Force, as algorithms developed by different researchers can be tested and compared using consistent simulations, collected data, and laboratory conditions.
2 INTRODUCTION

Radar systems are crucial for robust surveillance, target acquisition, and reconnaissance in all weather conditions and over wide ranges of interest. Anti-access/area denial (A2/AD) environments are especially challenging, therefore it is necessary to develop innovative signal and data processing techniques to provide advanced sensing capabilities to the warfighter. Distributed multiple input multiple output (MIMO) FAR systems provide the potential to exploit all available degrees of freedom on transmit and receive in order to maximize radar system performance, thus they offer much promise for improved sensing as well as the creation of new sensing modalities.

MIMO radar systems employ multiple transmit and receive elements with transmit elements that have the ability to transmit arbitrary waveforms simultaneously and receive elements that have the ability to process all of the transmitted signals jointly [1]-[10]. Of primary interest are MIMO ground moving target indicator (GMTI) systems consisting of distributed airborne platforms. As in traditional single input single output (SISO) and single input multiple output (SIMO) systems, effective clutter suppression is a key factor in system performance, and the availability of realistic clutter models and simulated clutter data is critical for advanced system design and performance analysis.

M&S tools developed by the Air Force include the Research Laboratory Space Time Adaptive Processing (RLSTAP) Algorithm Development Tool [11], which provides a flexible simulation and analysis tool for SIMO GMTI systems that employ monostatic space-time adaptive processing (STAP) [12]-[14], and the Signal Modeling and Simulation Tool for Multichannel Bistatic Systems (SMS-MBS) [15]-[17], which provides an advanced M&S capability for bistatic SIMO STAP systems. The SMS-MBS tool can generate multiple data cubes for multiple coherent processing intervals (CPIs) with automatically updated bistatic scenarios, but is limited to geometries for which the bistatic angle is small and the monostatic-bistatic equivalence theorem (MBET) approximation is valid. The MIMO Radar Clutter Modeling and Simulation (MIMO-CMS) tool [18],[19], developed by Metron, OSU, and Signal Processing Consultants, Inc. (SPC), provides a physics-based M&S capability for distributed airborne MIMO GMTI systems and extends the M&S capabilities to multistatic STAP [20] and multistatic MIMO [21] configurations. MIMO-CMS accurately characterizes the statistical and spectral properties of the clutter for a variety of radar operating parameters and site-specific geometry and scattering environments. It uses a physics-based bistatic scattering model developed by OSU and is not limited by the MBET approximation [22]. A companion set of toolbox functions calculates fundamental quantities such as iso-range contours, the radar range equation, mean clutter radar cross section (RCS), clutter amplitude and spectral characteristics, the true covariance matrix, and the clutter rank [23]. Additional functions perform MIMO signal processing, calculate performance metrics such as signal-to-noise-ratio (SNR), mean-square error (MSE), detection and false alarm probabilities; and perform statistical and experimental model goodness-of-fit tests. MIMO-CMS currently only provides a single CPI data cube for a fixed set of radar system parameters.

FAR systems, also called cognitive radar (CR) systems, mimic the PA cycle of cognition [24],[25] to adapt the radar sensor in real-time to collect data to achieve a required level of
performance [26]-[43]. This requires developing a perception of the current system status, prediction of the effect different sensing actions, and choosing the next sensing action, all in real-time. Haykin introduced the concept of cognitive radar in [26], however the motivation and many of the underlying ideas grew out of the fields of knowledge-aided radar signal processing [41]-[47], agile waveform design [48]-[55], sensor management [56]-[66], sonar ping control [67]-[69], bio-mimetic signal processing [70]-[75], and echoic flow [76],[77]. Although the concept of sensor adaptation goes back several decades, research into truly cognitive systems is currently in its infancy.

The potential of cognitive approaches to enhance existing radar performance in almost all respects has led to an upsurge in research in recent years. As of early 2015, the research had all been advancing concepts theoretically and examining their performance through simulation, or at best using pre-recorded data. There had been no reports of experimentally validated concepts, largely because the necessary hardware to test them had not been developed. However, this step is vital in order to establish the true performance potential of applying cognitive processing methods. To address this, the CSL at the OSU ESL, in conjunction with Metron, Inc., AFRL, and AFOSR, has embarked on a program of research to develop and examine cognitive radar processing concepts analytically and experimentally.

Under an AFOSR-sponsored grant from the 2013 Defense University Research Instrumentation Program (DURIP), the CSL designed and built the CREW, the world’s first radar testbed built specifically to allow testing of fully adaptive and cognitive algorithms. The CREW became operational in early 2015. Under AFRL-sponsored Small Business Innovative Research (SBIR) Topic AF 131-135 Fully Adaptive Radar, Metron and OSU have developed a theoretical framework for a cognitive FAR system that identifies and mathematically models the key system components within the context of target detection and tracking for a single sensor and target [30]-[34]. In the on-going FAR Phase II, we have been investigating theoretical extensions to the FAR framework and developing modeling, simulation, analysis, and experimentation capabilities to demonstrate the performance improvement achieved by FAR systems over traditional FFR systems [35]-[40]. We began with simulated scenarios [31]-[33] and pre-recorded data [34]-[35] from OSU’s SDR system [78]-[84]. Pre-recorded data offers limited opportunities to test cognitive algorithms since the sensing does not truly adapt in real-time, and cognitive processing must be done “after-the-fact” in an artificial manner. We now have the capability to demonstrate the real-time operation of the cognitive radar tracking system using the CREW [36]-[40]. Demonstrations were given to AFRL senior management in October 2015 and May 2016.

A key gap in the Air Force’s previously developed radar M&S tools is the lack of a comprehensive, dynamic distributed radar scenario generation capability for distributed FAR systems. Therefore, the goal of this project (SBIR Topic 161-132 FAR M&S Development) is to develop a MATLAB-based M&S architecture for distributed FAR radar that will enable algorithm development and testing on simulated, previously collected, and real-time streaming data. In Phase I, we have developed a baseline FAR M&S architecture that implements the FAR framework for a stationary sensor system. Throughout this report we use the term “FAR framework” to denote the mathematical model of a FAR system and the term “FAR M&S architecture” to denote the MATLAB-based software implementation of the FAR framework.
The FAR M&S architecture is coded in MATLAB using an OOP approach. It includes a FAR engine to control the operation of the PA cycle and software objects that determine the next set of sensing parameters; obtain data from the sensor; process the data to track the target; and store and display the results of the sensing and tracking processes. We have developed modules that implement simulated and pre-recorded data examples in [30]-[35], and the real-time CREW data examples in [36]-[39]. We have developed a simulation of the CREW and the application programming interface (API) layers for the simulated and experimental CREW data sources to enable switching between simulated and experimental data. A demonstration was given in March 2016 for members of the North Atlantic Treaty Organization (NATO) Sensors Electronics Technology (SET)-227 Panel on Cognitive Radar.

The FAR M&S architecture developed in Phase I allows for transparent switching between the simulated and experimental data sources, as well as between FAR algorithms that drive the sensing. The ability to easily interchange sensing and processing objects will allow for rapid development and testing of cognitive radar algorithms by structuring the M&S functions to avoid duplicating effort and “single point” solutions. It will enable collaboration between researchers in industry, academia, and the Air Force, as algorithms developed by different researchers can be tested and compared using consistent simulations, collected data, and laboratory conditions. In Phase II, we plan to make the FAR M&S architecture code available to members of the NATO SET-227 Panel on Cognitive Radar. Collaborations with members of this panel to develop and test algorithms on the CREW are already underway. Furthermore, several members have already begun development of their own cognitive radar test beds [85],[86] and our FAR M&S architecture will enable further collaboration within the panel using these data sources.

In Phase II, we also plan to extend the baseline architecture to model a dynamic, distributed airborne MIMO radar FAR system using the full MIMO-CMS tool as the simulation base. This will provide a comprehensive radar scenario generation capability that will fill a key gap in the Air Force’s previously developed radar M&S tools.

This report is organized as follows. In Chapter 3, we provide an overview of the FAR framework and the applications developed in [30]-[40]. In Section 3.1, we develop the theoretical framework and in Section 3.2, we give an overview of the applications. In Chapter 4, we provide a description of the FAR M&S architecture developed for this project. In Section 4.1 we describe the software structure and modules, in Section 4.2 we provide a guide to using the MATLAB code, in Section 4.3 we give a more detailed description of the base classes, and in Section 4.4, we provide a description of the CREW sensor object. Chapter 5 contains a summary and conclusions.
3 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Fully Adaptive Radar Framework

3.1.1 General Framework

The heart of cognition is the perception-action cycle that both informs and is informed by memory [24]-[25]. Cognition requires stimulation by sensors. In the human this is via hearing, touch, smell, vision, and taste. The nervous system processes the sensed stimuli and converts them into a perception of the world. We are able to take informed action by interpreting our perception of the world and making decisions. As a consequence, the nervous system sends signals that activate our muscles, thus enabling the desired action to take place. Informed decision-making is a key feature of the perception-action cycle. It requires the establishment of choices and the selection of one according to a desired goal. There is an element of prediction that arises in which perceptual information is combined with a model of the environment to predict the effect actions may have on it. Attention is closely related to perception and may be thought of as the requirement to allocate and direct the sensing resources towards relevant information.

Artificial cognition in a cognitive/FAR system attempts to mimic the PA cycle to make the best use of system resources for the situation at hand [26]-[29]. According to Haykin [28], a cognitive radar has the following capabilities: it predicts the consequences of actions, performs explicit decision-making, learns from the environment, and uses memory to store the learned knowledge. Haykin's approach uses Bayesian filtering [87],[88] as the basis for the artificial PA cycle. It provides the processing, perception, and memory inherent in a cognitive system. The Bayesian filter is augmented with a decision-making controller that uses perception from the filter and prediction of future outcomes to determine the next action taken by the sensor.

In [30]-[34], Metron and OSU developed a general mathematical framework for a cognitive/FAR system that incorporates the main components of cognition. The basic mathematical model of a cognitive sensor/processor system is shown in Figure 1. The system consists of four components: (i) the scene, which includes the target and the environment, (ii) the sensor that observes the scene and generally consists of a transmitter and receiver, (iii) the processor that converts the observed data into a perception of the scene, and (iv) the controller that determines the next actions taken by the sensor and processor based on feedback (perception) from the processor. The controller is the novel component that distinguishes a cognitive system from a traditional feed-forward sensor/processor system.

In this framework, we assume that the objective of the system is to estimate the state of a target in the scene. The target state at time $t_k$ is denoted as $x_k$. The sensor observes the scene and produces a measurement vector $z_k$ that depends on the target state $x_k$ and the sensor parameters $\theta_k$. We assume that the estimate of the target state at time $t_k$ is a function of the observations up to time $t_k$, which in turn depend on the sensor parameters up to time $t_k$, which we denote as $Z_k = \{z_1, z_2, \ldots, z_k\}$ and $\Theta_k = \{\theta_1, \theta_2, \ldots, \theta_k\}$, respectively.
We assume a first-order Markov motion model with initial target state probability density function \( q(x_0) \) and transition PDF \( q(x_k | x_{k-1}; \theta_k) \), which may depend on the sensor parameters. This will occur, for example, when the choice of sensor parameters affects the time difference \( t_k - t_{k-1} \). The measurement model is described by the conditional PDF \( f(z_k | x_k; \theta_k) \), which is also called the likelihood function. The cost of obtaining an observation and any constraints on the sensor parameters are modeled by the sensor cost function \( R_{\Theta}(\theta_k) \). The processor processes the data and produces an estimate of the target state \( \hat{x}_k(Z_k) \) by minimizing the expected value of the processor cost function \( C(\hat{x}_k(Z_k), x_k) \).

The controller decides on the next value for the sensor parameters \( \theta_k \) by minimizing a loss function \( L_{C,\Theta}(\cdot) \) that balances the performance of the processor via the processor cost function \( C(\cdot,\cdot) \) and the cost of using the sensor via the sensor cost function \( R_{\Theta}(\cdot) \).

For the first-order Markov motion model, the conditional or posterior PDF of \( x_k \) given \( Z_k \) may be obtained from the Bayes-Markov recursion [87]-[89]:

\[
\begin{align*}
  f^{-}(x_k) & \equiv f(x_k | Z_{k-1}; \Theta_k) = \int q(x_k | x_{k-1}, \theta_k) f^{+}(x_{k-1}) dx_{k-1} \\
  f^{+}(x_k) & \equiv f(x_k | Z_k; \Theta_k) = \frac{f(z_k | x_k; \theta_k) f^{-}(x_k)}{\int f(z_k | x_k; \theta_k) f^{-}(x_k) dx_k},
\end{align*}
\]

where \( f^{-}(x_k) \) is the motion-updated predicted density and \( f^{+}(x_k) \) is the information-updated posterior density. The recursion is initialized with

\[
f^{+}(x_0) = q(x_0).
\]
The conditional Bayes risk is the expected value of the processor cost function with respect to the conditional PDF of \( x_k \) given \( Z_k \),

\[
R^*_C (Z_k; \Theta_k) = E_k^+ \left\{ C(\hat{x}(Z_k), x_k) \right\},
\]

where \( E_k^+ \{ \cdot \} \) denotes expectation with respect to \( f^+ (x_k) \). The estimator is found by minimizing the conditional Bayes risk [90]:

\[
\hat{x}(Z_k) = \arg \min_{\hat{x}} R^*_C (Z_k; \Theta_k).
\]

In the controller, we assume that we have received the observations up to time \( t_{k-1} \) and want to find the next set of sensor parameters to optimize the performance of the state estimator that will include the next observation \( z_k \) as well as the previous observations \( Z_{k-1} \). We define the joint conditional PDF of \( x_k \) and \( z_k \) conditioned on \( Z_{k-1} \) as:

\[
f^+ (x_k, z_k) = f(x_k, z_k | Z_{k-1}; \Theta_k) = f(z_k | x_k; \theta_k) f(x_k | Z_{k-1}; \Theta_k)
\]

We define the predicted conditional Bayes risk for the estimator \( \hat{x}_k(Z_k) \) by taking the expectation of the processor cost function with respect to the joint conditional PDF \( f^+ (x_k, z_k) \),

\[
R^*_C (\theta_k | Z_{k-1}; \Theta_{k-1}) \equiv E_k^+ \left\{ C(\hat{x}(Z_k), x_k) \right\}.
\]

It is important to emphasize that the predicted conditional Bayes risk is a function of the known past observations \( Z_{k-1} \) but not the unknown next observation \( z_k \) since it has been averaged over both \( z_k \) and \( x_k \). It is also function of all the sensor parameters in \( \Theta_k \), however we separate the dependence on the unknown next sensor parameter \( \theta_k \) from the known past sensor parameters \( \Theta_{k-1} \) so that we may optimize over \( \theta_k \).

The next value of \( \theta_k \) is chosen to minimize a loss function that balances the predicted conditional Bayes risk and the sensor cost,

\[
L_{C,\theta} (\theta_k | Z_{k-1}; \Theta_{k-1}) = L \left\{ R^*_C (\theta_k | Z_{k-1}; \Theta_{k-1}), R_{\theta} (\theta_k) \right\}.
\]

The controller optimization problem is then given by:

\[
\theta_k = \arg \min_{\theta} L_{C,\theta} (\theta | Z_{k-1}; \Theta_{k-1}).
\]

The cognitive sensor/processor system framework described by Figure 1 and Eqs. (1)-(9) is very general and can be applied to many problems. It is based on the perception-action cycle and includes sensing, processing, perception, memory, attention, prediction, and decision-making. In [30]-[34], the framework was specialized for single target tracking, as well as for target detection and track initiation/termination. A summary of the results is given in Sections 3.1.2-3.1.4. By separating the general principles from the specific application and implementation details, our formulation provides a flexible framework applicable to the general tracking problem. It provides a generalization and formalism to the cognitive radar tracking formulations in [26]-[28],[51]-[65].
3.1.2 Cognitive Single Target Tracking

To specialize the framework for single target tracking, we specify the processor cost function and derive the corresponding state estimator and predicted conditional Bayes risk function used by the controller.

For vector parameter estimation, a commonly used cost function is the sum of the squared estimation errors, which is given by:

\[
C(\hat{x}(Z_k), x_k) = \text{tr}\left\{\left[\hat{x}(Z_k) - x_k\right]\left[\hat{x}(Z_k) - x_k\right]^T\right\}.
\]

For this cost function, the solution to (5) is the minimum mean-square error (MMSE) estimator, which is the conditional mean Error! Reference source not found.:

\[
\hat{x}(Z_k) = \mu_k^\tau = E_k^\tau \{x_k\}.
\]

The predicted conditional Bayes risk is given by:

\[
R_C^\tau (\theta_k | Z_{k-1}; \Theta_{k-1}) = \text{tr}\left\{S_k^\tau (\theta_k | Z_{k-1}; \Theta_{k-1})\right\},
\]

where

\[
S_k^\tau (\theta_k | Z_{k-1}; \Theta_{k-1}) = E_k^\tau \left\{\left[\hat{x}(Z_k) - x_k\right]\left[\hat{x}(Z_k) - x_k\right]^T\right\}
\]

is the predicted conditional MSE matrix.

In most cases, it is not possible to evaluate the MSE analytically or numerically. However, the Bayesian Cramér-Rao lower bound (BCRLB), which is the inverse of the Bayesian information matrix (BIM), provides a (matrix) lower bound on the MSE matrix of any estimator [90],[91] and is usually analytically tractable. Here we develop a predicted conditional BIM (PC-BIM), \(B_k^\tau (\theta_k | Z_{k-1}; \Theta_{k-1})\), and a predicted conditional Cramér-Rao lower bound (PC-CRLB) to bound the predicted conditional MSE matrix in (13),

\[
R_C^\tau (\theta_k | Z_{k-1}; \Theta_{k-1}) \geq \text{tr}\left\{C^T B_k^\tau (\theta_k | Z_{k-1}; \Theta_{k-1})^{-1} C\right\}.
\]

The PC-BIM may be expressed as the sum of two terms as follows:

\[
B_k^\tau (\theta_k | Z_{k-1}; \Theta_{k-1}) = B_k (\theta_k | Z_{k-1}; \Theta_{k-1}) + J_k (\theta_k | Z_{k-1}; \Theta_{k-1}).
\]

The first term is the predicted information matrix (PIM), which can be approximated by the inverse of the predicted covariance matrix:

\[
B_k (\theta_k | Z_{k-1}; \Theta_{k-1}) \approx \Sigma_k (\theta_k | Z_{k-1}; \Theta_{k-1})^{-1},
\]

and the second term is the expected value of the Fisher information matrix (FIM) with respect to the predicted density \(f^\tau (x_k)\). The expected Fisher information matrix (EFIM) is given by:

\[
J_k (\theta_k | Z_{k-1}; \Theta_{k-1}) = E_k (J_k (x_k; \theta_k)),
\]

where \(J_k (x_k; \theta_k)\) is the standard FIM [90],[91].

The expressions in (1)-(3), (11), and (14)-(17) provide the Bayes-Markov tracking recursion, the state estimate, and the predicted conditional Bayes risk expressions for a cognitive
sensor/processor system whose objective is single target tracking. The cognitive single target tracking system recursion is summarized in Table 1.

Table 1: Cognitive Single Target Tracking Recursion

<table>
<thead>
<tr>
<th>Initialization</th>
<th>( f^+ (x_0) = q(x_0) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller Optimization</td>
<td>( f^- (x_k; 0) = \int q(x_k</td>
</tr>
<tr>
<td></td>
<td>( B_k (\theta</td>
</tr>
<tr>
<td></td>
<td>( J_k (\theta</td>
</tr>
<tr>
<td></td>
<td>( B_k (\theta</td>
</tr>
<tr>
<td></td>
<td>( \theta_k = \arg \min_0 { L { B_k (\theta</td>
</tr>
<tr>
<td>Motion Update</td>
<td>( f^- (x_k) = \int q(x_k</td>
</tr>
<tr>
<td>Measurement</td>
<td>Obtain measurement ( z_k ) according to ( \theta_k )</td>
</tr>
<tr>
<td>Information Update</td>
<td>( f^+ (x_k) = \frac{f(z_k</td>
</tr>
<tr>
<td>Track Estimate</td>
<td>Obtain ( \hat{x}(Z_k) ) from mean of ( f^+(x_k) )</td>
</tr>
</tbody>
</table>

3.1.3 Cognitive Single Target Tracking, Initiation, and Termination

For track initiation and termination, the system objective is to minimize the time to detect the presence or absence of a target. We do this by maximizing the probability of making a correct decision at each time. To specialize the general cognitive framework for this problem, we follow the likelihood ratio detection and tracking (LRDT) methodology in Chapter 7 of [87]. On \( H_1 \) when the target is present, \( x_k \in X \), where \( X \) is the target-present state space. We define a null (target-absent) state \( \emptyset \) so that on \( H_0 \) when the target is absent, \( x_k = \emptyset \):

\[
H_0 : x_k = \emptyset \\
H_1 : x_k \in X.
\]
We define the augmented state space $X^\varnothing \equiv X \cup \varnothing$ and develop the Bayes-Markov recursions for this model. We then specify the processor cost function and derive the corresponding state estimator and predicted conditional Bayes risk function used by the controller.

A PDF on the augmented state space $X^\varnothing$ is characterized by two components $P(\varnothing)$ and $f(\mathbf{x}_k)$, where

$$P(\varnothing) \equiv \Pr(\mathbf{x}_k = \varnothing),$$

and $f(\mathbf{x}_k)$ is the conditional PDF of $\mathbf{x}_k$ given that $\mathbf{x}_k \notin X$. Therefore, the predicted and posterior PDFs on the augmented state space are characterized by components $P^-(\varnothing)$, $f^-(\mathbf{x}_k)$ and $P^+(\varnothing)$, $f^+(\mathbf{x}_k)$, respectively.

The initial target PDF is characterized by $P(\varnothing)$ and $q(\mathbf{x}_0)$ and the transition density on the augmented state space is characterized by four components:

$$P(\varnothing_k | \varnothing_{k-1}) \equiv \Pr(\mathbf{x}_k = \varnothing | \mathbf{x}_{k-1} = \varnothing) \quad \mathbf{x}_k = \varnothing, \mathbf{x}_{k-1} = \varnothing$$
$$P(\varnothing_k | \mathbf{X}_{k-1}) \equiv \Pr(\mathbf{x}_k = \varnothing | \mathbf{x}_{k-1} \in \mathbf{X}) \quad \mathbf{x}_k = \varnothing, \mathbf{x}_{k-1} \in \mathbf{X}$$
$$q(\mathbf{x}_k | \varnothing_{k-1}) \quad \mathbf{x}_k \in \mathbf{X}, \mathbf{x}_{k-1} = \varnothing$$
$$q(\mathbf{x}_k | \mathbf{x}_{k-1}; \theta_k) \quad \mathbf{x}_k \in \mathbf{X}, \mathbf{x}_{k-1} \in \mathbf{X}.$$

The likelihood function on the augmented state space is characterized by the likelihood functions on $H_0$ and $H_1$, which are denoted by $f(\mathbf{z}_k | \varnothing_k; \theta_k)$ and $f(\mathbf{z}_k | \mathbf{x}_k; \theta_k)$, respectively.

We express the Bayes-Markov recursions for the predicted and posterior PDFs as recursions on $P^-(\varnothing_k)$, $f^-(\mathbf{x}_k)$, $P^+(\varnothing_k)$, and $f^+(\mathbf{x}_k)$. Following [87], the Bayes-Markov recursions are initialized with:

$$P^+(\varnothing) = P(\varnothing)$$

$$f^+(\mathbf{x}_0) = q(\mathbf{x}_0).$$

If we assume that the transition probabilities are defined such that the motion update does not affect the probability of being in the null state, then we have:

$$P^-(\varnothing_k) = P^+(\varnothing_{k-1}).$$

Under this assumption, we obtain the “simplified recursion” [87] in which the predicted PDF on $H_1$ is found from:

$$f^-(\mathbf{x}_k) = P(\varnothing_k | \mathbf{X}_{k-1})q(\mathbf{x}_k | \varnothing_{k-1}) \left[1 - P(\varnothing_k | \mathbf{X}_{k-1})\right] \int q(\mathbf{x}_k | \mathbf{x}_{k-1}; \theta_k) f^+(\mathbf{x}_{k-1}) d\mathbf{x}_{k-1}. \tag{24}$$

For target detection, we assume that the state estimate takes one of two values, $\hat{\mathbf{x}}_k(\mathbf{Z}_k) = \varnothing$ or $\hat{\mathbf{x}}_k(\mathbf{Z}_k) \cap \mathbf{X}$, thus the estimation problem becomes a binary detection problem. We assume the standard binary detection cost function [87],[90], and the optimal state estimator (decision rule), found by minimizing the conditional Bayes risk, is the Bayesian integrated likelihood ratio test (BLRT) [87]:

Approved for public release; distribution is unlimited
\[ \begin{align*} 
\hat{x}_k \in X \\
\bar{\Lambda}(Z_k) \geq \tau, \\
\hat{x}_k = \emptyset 
\end{align*} \] (25)

where \( \bar{\Lambda}(Z_k) \) is the \textit{Bayesian integrated likelihood ratio} (BLR). For the simplified recursion, it has the form:

\[ \bar{\Lambda}(Z_k) \equiv \bar{\Lambda}(Z_{k-1}) \bar{\mathcal{L}}(z_k | Z_{k-1}), \] (26)

where \( \bar{\mathcal{L}}(z_k | Z_{k-1}) \) is the \textit{integrated likelihood ratio} (ILR) for the current data, defined as:

\[ \bar{\mathcal{L}}(z_k | Z_{k-1}) \equiv \int \frac{f(z_k | x_k; \theta_k)}{f(z_k | \emptyset; \theta_k)} f^{-}(x_k) dx_k. \] (27)

The information update is given by:

\[ P^+(\emptyset_k) = \frac{1}{1 + \bar{\Lambda}(Z_k)} \] (28)

\[ f^+(x_k) = \frac{f(z_k | x_k; \theta_k) f^{-}(x_k)}{\int f(z_k | x_k; \theta_k) f^{-}(x_k) dx_k}. \] (29)

As discussed in [30],[32],[33], the condition in (23) can be satisfied if we set

\[ P(\emptyset_k | X_{k-1}) = \min \left\{ \bar{\Lambda}(Z_{k-1})^{-1}, P_{\emptyset} \right\}, \] (30)

where \( P_{\emptyset} \) is a fixed upper limit close to one.

The predicted conditional Bayes risk includes probabilities of missed and false detections, which are generally difficult to calculate, so we must resort to an approximation or surrogate function to perform the controller optimization. As discussed in [30],[32],[33], we take a heuristic approach and use the same criterion that we used for track estimation, namely to minimize the trace of the PC-CRLB, which is the inverse of the PC-BIM. This maximizes the Bayesian information, which is useful for making statistical inferences about \( x_k \) in both the estimation and detection settings. Thus,

\[ R^c \left( \theta_k | Z_{k-1}; \Theta_{k-1} \right) \Rightarrow \text{tr} \left\{ C^T B^k_z (\theta_k | Z_{k-1}; \Theta_{k-1})^{-1} C \right\}, \] (31)

where we use the notation \( \Rightarrow \) to denote replacement by a surrogate function.

For track initiation, we initially assume the target is absent and initialize \( \bar{\Lambda}(Z_0) \) to some small value well below the target-present detection threshold \( \tau_P \). If a target is present, then the BLR will grow over time as evidence in favor of \( H_1 \) is accumulated. Eventually it will cross the threshold and the target will be declared present. If the target is absent, the BLR will decrease over time. However, if we allow the BLR to decrease below the initial value during periods when there is no target, then when a target appears and the BLR starts to grow, it will take longer to detect. To prevent this, we restrict the BLR to stay at or above the initial value, which we denote as \( \bar{\Lambda}_{\min} \).

Once the target is declared present, processing continues in the same manner, except now we assume the target is present and restrict the BLR to stay at or below some maximum value.
\( \bar{A}_{\text{max}} \), which is well above the target-absent detection threshold \( \tau_d \). To allow some robustness in the system, we set \( \tau_d \) to be less than \( \tau_P \). This way, once the BLR exceeds \( \tau_P \) and the target is declared present, sufficient evidence in favor of \( H_0 \) must accumulate before the BLR drops below \( \tau_d \) and the target is declared absent again.\(^1\) Since the target is assumed present, we can compute a target state estimate after the information update and we obtain the simultaneous single target tracking and track initiation/termination recursion summarized in Table 2. This recursion reduces to the single target tracking recursion in Table 1 if we always set \( P(\emptyset_k | X_{k-1}) = 0 \) on line 5. This recursion is used in the distributed sensor resource allocation examples in [30]-[34].

\(^1\) Proper operation of the track initiation and termination recursions requires that \( t_r \leq L_{\text{max}} \) and \( t_d \geq L_{\text{min}} \).
Table 2: Cognitive Single Target Tracking, Initiation, and Termination Recursion

<table>
<thead>
<tr>
<th>Initialization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Declare target absent</td>
<td></td>
</tr>
<tr>
<td>2 $\overline{\Lambda}(Z_0) = \overline{\Lambda}_{\min}$</td>
<td></td>
</tr>
<tr>
<td>3 $P^*(\emptyset_0) = (1 + \overline{\Lambda}(Z_0))^{-1}$</td>
<td></td>
</tr>
<tr>
<td>4 $f^+(x_0) = q(x_0)$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Motion Update – Part I</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5 $P(\emptyset</td>
<td>X_{k-1}) = \min{\overline{\Lambda}(X_{k-1})^{-1}, \overline{\Lambda}(\emptyset)}$</td>
</tr>
<tr>
<td>6 $P^<em>(\emptyset_k) = P^</em>(\emptyset_{k-1})$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controller Optimization</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7 $f^-(x_k; \theta) = P(\emptyset</td>
<td>X_{k-1}) q(x_k</td>
</tr>
<tr>
<td>8 $B_k^*(\theta</td>
<td>Z_{k-1}; \Theta_{k-1}) = \Sigma_k^{-1}(\theta</td>
</tr>
<tr>
<td>9 $J_k^*(\theta</td>
<td>Z_{k-1}; \Theta_{k-1}) = E_k^{-1}(J(x_k; \theta))$</td>
</tr>
<tr>
<td>10 $B_k^*(\theta</td>
<td>Z_{k-1}; \Theta_{k-1}) = B_k^*(\theta</td>
</tr>
<tr>
<td>11 $\theta_k = \arg \min_0 L{B_k^*(\theta</td>
<td>Z_{k-1}; \Theta_{k-1}), R_0(\theta)}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Motion Update – Part II</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12 $f^-(x_k) = P(\emptyset</td>
<td>X_{k-1}) q(x_k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measurement</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>13 Obtain measurement $z_k$ according to $\theta_k$</td>
<td></td>
</tr>
</tbody>
</table>

BLR

<table>
<thead>
<tr>
<th>Information Update</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>14 $P^*(\emptyset_k) = (1 + \overline{\Lambda}(\emptyset_k))^{-1}$</td>
<td></td>
</tr>
<tr>
<td>15 $f^+(x_k) = \overline{\Lambda}(x_k; \theta_k) f^+(x_k) dx_k$</td>
<td></td>
</tr>
</tbody>
</table>

BLRT

<table>
<thead>
<tr>
<th>Track Estimate (if target present)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16 if target absent and $\overline{\Lambda}(Z_k) \geq \tau_p$, then declare target present and initiate track, elseif target present and $\overline{\Lambda}(Z_k) &lt; \tau_A$, then declare target absent and terminate track</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Track Estimate (if target present)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15 Obtain $\hat{x}(Z_k)$ from mean of $f^+(x_k)$</td>
<td></td>
</tr>
</tbody>
</table>
3.1.4 Cognitive MAP-PF Single Target Tracking

The recursions in Tables 1 and 2 are applicable to detection-based and track-before-detect tracking systems. In a detection-based system, the sensor performs “hard detection,” i.e. there is some signal processing of the sensor data that converts the data to a detection surface, which is then thresholded to produce a “detection” in the form of a measurement in the natural measurement space of the sensor (e.g. angle, range, and/or Doppler). The observation \(z_k\) is the measurement obtained from hard detection processing. In a track-before-detect system, \(z_k\) is the sensor data and the information update involves computing the likelihood function of the sensor data with respect to the target state vector. This can be computationally intensive but can yield significant performance improvements over a detection-based system. In this subsection, we extend the recursions to maximum a posteriori penalty function (MAP-PF) tracking systems. The MAP-PF methodology applies to problems in which the likelihood function \(f(z_k | x_k; \theta_k)\) depends on the target state vector only through a known, possibly nonlinear mapping to the natural measurement space of the sensor. The MAP-PF methodology offers reduced computational complexity over track-before-detect systems, while maintaining the performance advantage over detection-based systems. The cognitive MAP-PF tracking system, in which the processor includes the detector and tracker, is shown in Figure 2. It is particularly suitable for FAR systems, as it already contains feedback within the tracking processor.

MAP-PF is a multi-target tracking methodology developed in [92]-[97] and described in Chapter 6 of [87]. In this approach, the multi-target track estimation problem is formulated directly from the sensor data \(z_k\) using the maximum a posteriori (MAP) estimation criterion. The penalty function method of nonlinear programming [98] is used to obtain a tractable solution. The result is a two-step estimation process similar to traditional feed-forward detection-based

![Figure 2: Cognitive MAP-PF Sensor/Processor System Framework](image)
systems, except the processes are coupled via the penalty function and the data association step of traditional multi-target tracking approaches is eliminated. In the detection process, the penalty function uses the current target states to guide the detector to the relevant region of the detection surface. In the track estimation process, the penalty function determines the influence of the detector measurements on the final track estimates by adaptively adjusting the measurement error variance using the FIM.

Let $y_k$ denote the natural parameters. They are related to the state parameters by the nonlinear mapping

$$ y_k = h(x_k). \quad (32) $$

Let $f(z_k | y_k; \theta_k)$ denote the likelihood function with respect to the natural parameters and let $J_y(y_k; \theta_k)$ denote the FIM of the natural parameters. The likelihood function and FIM with respect to the state parameters are given by [90]:

$$ f(z_k | x_k; \theta_k) = f(z_k | y_k = h(x_k); \theta_k) \quad (33) $$

$$ J_x(x_k; \theta_k) = H(x_k)^T J_y(h(x_k); \theta_k) H(x_k), \quad (34) $$

where $H(x_k)$ is the Jacobian matrix, defined as:

$$ H(x_k) \equiv \left[ \nabla_x h(x) \right]^T \bigg|_{x=x_k}. \quad (35) $$

The MAP-PF algorithm employs a penalty function $\phi(y, h(x))$ which is equal to zero when $y = h(x)$ and becomes smaller (more negative) as the distance between $y$ and $h(x)$ increases. For example, a quadratic penalty function is:

$$ \phi(y, h(x); \Omega) \equiv -\frac{1}{2} [y - h(x)]^T \Omega^{-1} [y - h(x)], \quad (36) $$

where $\Omega$ is a matrix chosen to weight the components of the penalty function in some desirable manner. The MAP-PF motion update is the same as in the standard Bayes-Markov recursion and the MAP-PF information update is given by:

$$ \hat{x}_k = \arg\max_{x_k} f^-(x_k) \quad (37) $$

$$ \hat{y}_k = \arg\max_y \ln f(z_k | y; \theta_k) + \phi(y, h(\hat{x}_k); \Omega) \quad (38) $$

$$ f^-(x_k) = \frac{\exp\{\phi(\hat{y}_k, h(x_k); \Omega)\}}{\int \exp\{\phi(\hat{y}_k, h(x_k); \Omega)\} f^-(x_k) dx_k}. \quad (39) $$

In the first step, the MAP estimate of the predicted density is found. Depending on the implementation, it may be easier to find the MMSE estimate, which is the mean of the predicted density, instead.

In the second step, the optimization problem in (38) is a penalized maximum likelihood (ML) problem. If the second term in (38) had the form $\ln f(y)$, it would be a MAP estimation.
problem. Thus the penalty function can be interpreted as a prior term in a MAP estimation problem. In a traditional feed-forward detection-based tracking system, the optimal detector would solve the standard ML problem (i.e. (38) without the penalty function) to get the detector measurement. In MAP-PF, the penalty function restricts the detector estimate to be in the vicinity of where the tracker predicts it to be, hence MAP-PF is performing “guided” detection. By specifying a quadratic penalty function, we are implicitly modeling the prior distribution of $y_k$ as Gaussian with mean $h(\hat{x}_k)$ and covariance matrix $\Omega$. We have some flexibility in choosing $\Omega$, and a logical choice would be the covariance matrix of the predicted density of $y_k$ obtained from a transformation of the predicted density $f^-(x_k)$. Using a locally linear approximation of the function $h(x)$ at the point $\hat{x}_k$, we choose $\Omega$ to be the predicted covariance matrix of $y_k$, which is approximately given by:

$$\Omega_{MAP,k} = \Sigma_{y_k}^{-1} \approx H(\hat{x}_k) \Sigma_{y_k}^{-1} H(\hat{x}_k)^T.$$  \hspace{1cm} (40)

The third step in (39) looks like a standard information update with $\hat{y}_k$ acting as the measurement vector and the exponential of the penalty function, $\exp\left\{\phi(\hat{y}_k; h(x_k); \Omega)\right\}$, acting as the measurement likelihood function $f(\hat{y}_k | x_k; \theta_k)$. Here the quadratic penalty function is implicitly modeling $\hat{y}_k$ as Gaussian with mean $h(x_k)$ and covariance matrix $\Omega$. As in [87],[92]-[97], we choose $\Omega$ to be the inverse of the FIM of the natural parameters, $J_y(y_k; \theta_k)$. Calculation of the FIM often requires knowledge of the true value of $y_k$, however we can obtain a reasonably accurate approximation to the FIM by substituting in an estimate of $y_k$. The transformation of the predicted state estimate $h(\hat{x}_k)$ is a less volatile estimate than the current measurement $\hat{y}_k$, therefore we evaluate the FIM at $h(\hat{x}_k)$. Thus, for the information update we choose

$$\Omega_{I,k}^{-1} = J_y\left(h(\hat{x}_k); \theta_k\right).$$ \hspace{1cm} (41)

The MAP-PF single target tracking recursion is summarized in Table 3. We also developed a MAP-PF single target tracking, initiation, and termination recursion in [30]. The recursion in Table 3 is used in the SDR and CREW examples in [30],[34]-[40].
<table>
<thead>
<tr>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f^+ (x_0) = q(x_0)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Controller Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f^- (x_k; \theta) = \int q(x_k</td>
</tr>
<tr>
<td>$B_k (0</td>
</tr>
<tr>
<td>$J_k (0</td>
</tr>
<tr>
<td>$B_k^* (0</td>
</tr>
<tr>
<td>$\theta_k = \arg \min_\theta L {B_k^* (0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Motion Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f^- (x_k) = \int q(x_k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain measurement $z_k$ according to $\theta_k$</td>
</tr>
<tr>
<td>Obtain $\hat{x}_k^-$ from mean or maximum of $f^- (x_k)$</td>
</tr>
<tr>
<td>$\Omega_{MAP,k} = H(\hat{x}_k^-) \Sigma_k H(\hat{x}_k^-)^T$</td>
</tr>
<tr>
<td>$\hat{y}_k = \arg \max_y \ln f(z_k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Information Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Omega^{-1}_{I,k} = J_y (h(\hat{x}_k^-); \theta_k)$</td>
</tr>
<tr>
<td>$f^+ (x_k) = \frac{\exp {\phi(\hat{y}<em>k, h(x_k); \Omega</em>{I,k})} f^- (x_k)}{\int \exp {\phi(\hat{y}<em>k, h(x_k); \Omega</em>{I,k})} f^- (x_k) dx_k}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Track Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obtain $\hat{x}(Z_k)$ from mean or maximum of $f^+ (x_k)$</td>
</tr>
</tbody>
</table>
3.2 Examples

For a specific application, we need to specify the components of the state vector, the motion and measurement models, the sensor parameters being controlled, and the form of the controller loss function. Finally, we need to specify the implementation details that include the type of tracker used to implement the Bayes-Markov recursion and the method for solving the controller optimization problem. In [30]-[34], we showed how the general tracking framework could be specialized for a distributed sensor system similar to the cognitive radar networks in [64],[65], in which system resources (observation time on each sensor) were allocated to optimize tracking performance. Using simulated data, we showed that the cognitive radar system offered significant performance gains over a standard feed-forward radar system. In [30],[34],[35], we showed how the tracking framework could be applied to a single sensor pulse-Doppler radar system in which the pulse repetition frequency (PRF) is adjusted to optimize tracking performance, while keeping the target from being Doppler-aliased and away from the zero-Doppler clutter. Results were shown on experimentally collected data using OSU’s SDR system. In [36]-[39], we applied the same algorithm to real-time data using OSU’s CREW system, and also allowed for simultaneous adjustment of the PRF and number of pulses.

3.3 Summary

In this section, we provided an overview of the general framework for a cognitive sensor/processor tracking system developed in [30]-[34]. The framework is based on the perception-action cycle and includes sensing in the sensor; processing in the detector and tracker; perception in the conversion of sensor data to the posterior PDF of the state vector; memory of all the past data in the posterior PDF; attention in the penalty function of the guided adaptive detector, which focuses the detector on the relevant region of the detection surface; prediction in the PC-BIM, which predicts the performance of the next measurement; and decision-making in the controller, which decides on the next values for the sensor parameters based on the predicted performance.
4 RESULTS AND DISCUSSION

4.1 FAR M&S Software Architecture Overview

The FAR M&S codebase has been designed in an object-oriented architecture so that the interfaces for the various components (optimizer, sensor, processor, etc.) are defined by base classes. Specific implementations of these objects (for example, an optimizer for five parameters) are subclasses of their respective base class so that they inherit common methods and properties.

The FAR M&S architecture is managed by a FAR_Engine. A conceptual block diagram of the FAR_Engine showing the basic objects, processing, and data flow is shown in Figure 3. The FAR_Engine consists of eight objects:
1. Scene [Simulation only]: defines the target, clutter and noise characteristics used for simulation.
2. Optimizer: solves the controller optimization problem to obtain the next set of sensor parameters
3. Sensor: obtains raw data from simulation, pre-recorded data, or experimentation.
4. Processor: performs raw data processing and Bayesian filtering.
5. StorageManager: maintains a history of variables of interest from the Scene, Optimizer, Sensor, and Processor and stores them to a file (long-term memory).
6. DisplayManager: displays values of interest each cycle while the algorithm is running and provides a final display of quantities of interest.
7. TimingManager: maintains timing during the cycle.
8. PerceptionActionCycle: runs one cycle of the PA cycle by calling object methods (functions) in the proper sequence, as shown in Figure 3.

The repository where the codebase is stored has been divided into two folders: an “architecture” folder to store the base classes and other common pieces of code, and a “modules” folder which stores the specific subclasses or implementations. The “architecture” folder contains eight files defining the base classes plus a UtilityFunctions object, which contains some commonly used functions. The FAR Engine is written as a script and there is no base class. The files in the “architecture” folder provide the interfaces (that is, common properties and methods) for how to create specific objects that provide functionality to different parts of the PerceptionActionCycle and FAR_Engine. The files in this folder should not be modified.

In Phase I, we developed specific implementations of the FAR M&S architecture for three examples: (i) the distributed sensor resource allocation (DSRA) simulation example in [33], (ii) the SDR pre-recorded data example in [35], and (iii) a CREW example in which five parameters are optimized and the data may be from simulation or real time experimentation. The “modules” folder contains separate folders for each of these examples named “DSRA,” “SDR,” and “CREW.”
Figure 3: FAR M&S Architecture Block Diagram
4.2 Using the FAR M&S MATLAB Code

4.2.1 System Requirements

To run the codebase, you will need MATLAB version 2014b or later. It is possible that it will run on earlier versions, but this has not been tested. No MATLAB Toolboxes are required.

4.2.2 Getting Started

To get up and running, do the following:

- Install the codebase into a folder on your computer.
- Start MATLAB.
- In MATLAB, navigate to the folder that you just stored. In this folder there should at least be a “code” folder and a file named `set_matlab_path_far_codebase()`.
- At the command prompt, type `set_matlab_path_far_codebase()`. This sets the proper paths for use with the code.

You can now run any of the three examples provided in the codebase. Navigate to one of the folders “code/modules/DSRA”, “code/modules/SDR”, or “code/modules/CREW”. Open the FAR_Engine file for that example (`FAR_Engine_JSTSP`, `FAR_Engine_SDR_PRF`, or `FAR_Engine_CREW_5Par`) and run it.
4.3 FAR M&S Architecture Base Classes

In this section, we provide an overview of the base classes in the FAR M&S Architecture. For the Scene, Sensor, Processor, Optimizer, Storage Manager, Display Manager, and Timing Manager, we provide a list of properties and abstract methods in each class. The PerceptionActionCycle class implements a method to run the cycle and it is described in detail.

The DSRA, SDR, and CREW modules provide implementations of each of the base classes and the FAR_Engines to run the examples for those applications. In Section 4.4, we provide a description of the CREW Sensor and Scene objects. We do not provide descriptions of the remaining codebase modules.

4.3.1 Scene

The Scene base class consists of
• Properties: N/A
• Abstract Methods: N/A

This is currently just a placeholder as we have not identified properties or methods that are common to all Scenes.

4.3.2 Sensor

The Sensor base class consists of:
• Properties: N/A
• Abstract Methods: get_measurement

4.3.3 Processor

The Processor base class consists of:
• Properties: MotionModel (structure), MeasModel (structure), Settings (structure)
• Abstract Methods: preprocess_rawdata, prior_initialization, predicted_update, posterior_update

4.3.4 Optimizer

The Optimizer base class consists of:
• Properties: PredActualFlag
• Abstract Methods: get_sensor_parameters, set_sensor_params, opt_sensor_params_pred_update

4.3.5 Storage Manager

The StorageManager base class consists of:
• Properties: Params (structure), history (structure)
• Abstract Methods: history_update
4.3.6 Display Manager

The DisplayManagerManager base class consists of:

- Properties: Params (structure), DisplayData (structure)
- Abstract Methods: store_display_data, display_cycle

4.3.7 Timing Manager

The TimingManager base class consists of:

- Properties: time (structure), k
- Abstract Methods: cycle_timing_init, cycle_timing_opt, cycle_timing_sens,
  cycle_timing_proc, cycle_timing_mgmt, timing_update

4.3.8 Perception Action Cycle

The PerceptionActionCycle base class consists of:

- Properties: f_post (structure), theta_opt (structure), cycle_timing (structure)
- Methods: PerceptionActionCycle (constructor), run_cycle

The PerceptionActionCycle base class provides implementations for the object
constructor method and for the run_cycle method. Thus, the PerceptionActionCycle object is
common to all examples.

The properties f_post (the posterior density), theta_opt (the optimum sensor parameters
for the current cycle), and cycle_timing (various timing values for the current cycle) are updated
during the execution of run_cycle and are passed out to the FAR_Engine at the end of the cycle.
They are passed back in to run_cycle at the beginning of the next cycle. Passing these
parameters to the next cycle is represented by the purple short term memory data in Figure 3.

The constructor method PerceptionActionCycle(f_post, theta0) is called by FAR_Engine
up front to create the PerceptionActionCycle object and initialize the posterior density and sensor
parameters. The code is

```matlab
function cycleObj = PerceptionActionCycle(f_post, theta0)

if nargin >0
    cycleObj.f_post       = f_post;
    cycleObj.theta_opt    = theta0;
    cycleObj.cycle_timing = [];
end % if nargin

end % constructor function
```

The run_cycle method implements the PA cycle. A detailed description is provided
below.
function [cycleObj, sensorObj, processorObj, timingObj, storageObj, displayObj]...
    = run_cycle(cycleObj, sceneObj, sensorObj, processorObj, optimizerObj, timingObj, storageObj, displayObj)

This defines the interface for calling run_cycle. Inputs are the scene, sensor, processor, optimizer, timing, storage, and display objects. During run_cycle, the cycle, sensor, processor, timing, storage, and display objects are updated and passed out to the FAR_Engine.

%------------------------------------------------------------------
% previous sensor parameters
%------------------------------------------------------------------
theta_old = cycleObj.theta_opt;

This gets the optimum sensor values from the last cycle.

%------------------------------------------------------------------
% initialize cycle timing
%------------------------------------------------------------------
cycleObj.cycle_timing = timingObj.cycle_timing_init;

This calls the method cycle_timing_init, implemented in timingObj, a specific implementation of the TimingManager object. It initializes the timing values for the current cycle.

%------------------------------------------------------------------
% tentative motion update & theta optimization
%------------------------------------------------------------------
[cycleObj.theta_opt, optParams, f_pred] = ...
    optimizerObj.opt_sensor_params_pred_update(...
        theta_old, cycleObj.f_post, sensorObj, processorObj,...
        cycleObj.cycle_timing);

This calls the method opt_sensor_params_pred_update, implemented in optimizerObj, a specific implementation of the Optimizer object. It performs the controller optimization to obtain the next set of sensor parameters. Inputs are the previous set of sensor values, the posterior density from last cycle, the sensor and processor objects, and the cycle_timing structure. As part of the optimization, the predicted density is computed for each set of sensor values that is evaluated. In some cases, the actual predicted density is computed and returned to the PA cycle in the variable f_pred. In this case the property optimizerObj.PredActualFlag is set equal to true. If not, it is false. The optimum sensor parameters are returned in cycleObj.theta_opt and other quantities of interest calculated during the optimization are returned in optParams. These will be stored or displayed later.

    cycleObj.cycle_timing = timingObj.cycle_timing_opt(optimizerObj,...
        cycleObj.cycle_timing);

This calls the method cycle_timing_opt, implemented in timingObj. It stores the optimization timing values for the current cycle. Inputs are the optimizer object and the cycle timing structure, and the output is the updated cycle timing structure.
% set sensor/processor parameters and get measurement

[sensorObj, processorObj] = optimizerObj.set_sensor_params(...
cycleObj.theta_opt, sensorObj, processorObj);

This calls the method set_sensor_params, implemented in optimizerObj. It updates the sensor and processor objects with the new sensor parameter values. Inputs are the optimum sensor values and the sensor and processor objects. Outputs are the updated sensor and processor objects.

[rawdata, sens_timing] = sensorObj.get_measurement(sceneObj,...
cycleObj.cycle_timing);

This calls the method get_measurement, implemented in sensorObj, a specific implementation of the Sensor object. It gets the measurement for the appropriate data source. Inputs are the scene object (if simulation), and timing for the data collect. Outputs are the raw sensor data and sensor timing values.

cycleObj.cycle_timing = timingObj.cycle_timing_sens(sens_timing,...
cycleObj.cycle_timing);

This calls the method cycle_timing_sens, implemented in timingObj. It stores the sensor timing values for the current cycle. Inputs are the sensor timing structure and the cycle timing structure, and the output is the updated cycle timing structure. One of the values is a flag that indicates if data was available. The following loop to process the data is only executed if data is available.

if cycleObj.cycle_timing.data_available

% actual motion update

if ~optimizerObj.PredActualFlag
    f_pred = processorObj.predicted_update(cycleObj.f_post, ...
cycleObj.cycle_timing.dt, cycleObj.theta_opt, theta_old);
end

This calls the method predicted_update, implemented in processorObj, a specific implementation of the Processor object. It performs the Bayesian filtering motion update, if not already performed as part of the sensor optimization. Inputs are the posterior density from last cycle, the total scan time of the current cycle, and the current and previous values of the sensor parameters. The output is the predicted density.

% pre-process raw data

zdata = processorObj.preprocess_rawdata(rawdata, sensorObj);

This calls the method process_rawdata, implemented in processorObj. It performs pre-processing of the raw data to get it into the form required by the information update, such as
converting raw radar data to a range/Doppler surface. Inputs are the raw data and the sensor object. The output is the processed data.

%--------------------------------------------------------------------
% information update
%--------------------------------------------------------------------
[cycleObj.f_post, LF, proc_timing] = ...
    processorObj.posterior_update(f_pred, zdata);

This calls the method posterior_update, implemented in processorObj. It performs the information update. Inputs are the predicted density and processed data. Outputs are the posterior density, the likelihood function structure containing quantities of interest to be stored or displayed later, and processor timing values.

else
    f_pred = cycleObj.f_post;
    LF = [];
    proc_timing.track_new = 0;
end % if data_available

If there is no data available, the predicted density is the posterior density from the last cycle, and the posterior density is also the posterior density from last cycle. The likelihood structure is empty and the flag indicating a new track is false.

cycleObj.cycle_timing = timingObj.cycle_timing_proc(proc_timing,...
    cycleObj.cycle_timing);

This calls the method cycle_timing_proc, implemented in timingObj. It stores the processor timing values for the current cycle. Inputs are the processor timing structure and the cycle timing structure, and the output is the updated cycle timing structure.

%--------------------------------------------------------------------
% storage update
%--------------------------------------------------------------------
storageObj = storageObj.history_update(cycleObj.cycle_timing,...
    cycleObj.theta_opt, optParams, f_pred, cycleObj.f_post, LF);

This calls the method history_update, implemented in storageObj, a specific implementation of the StorageManager object. It stores quantities of interest from the current cycle in the history structure, to be saved to a file later. Inputs are the cycle timing, the optimum sensor parameters, the additional optimization values, the predicted density, the posterior density, and the likelihood function structure. The output is the updated storage object.

%--------------------------------------------------------------------
% display update
%--------------------------------------------------------------------
if displayObj.Params.PlotCycleFlag
    displayObj = displayObj.storeDisplayData(cycleObj.cycle_timing,...
        cycleObj.theta_opt, optParams, f_pred, cycleObj.f_post, LF);
    displayObj.display_cycle(timingObj.k);
end
There is an option not to display any data while the algorithm is running in order to save time. If the displayObj.Params.PlotCycleFlag is false, then nothing is done. If it is true, then this code calls the methods store_display_data and display_cycle, implemented in displayObj, a specific implementation of the DisplayManager object. The method store_display_data stores quantities of interest from the current cycle in the display object and display_cycle updates the display screen. Inputs to store_display_data are the cycle timing, the optimum sensor parameters, the additional optimization values, the predicted density, the posterior density, and the likelihood function structure. The output is the updated display object. The inputs to display_cycle are the current cycle index and (implicitly) the display object.

```
cycleObj.cycle_timing = timingObj.cycle_timing_mgmt(...
    cycleObj.cycle_timing);
```

This calls the method cycle_timing_mgmt, implemented in timingObj. It stores the storage and display management timing values for the current cycle. The input is the cycle timing structure, and the output is the updated cycle timing structure.

```
%--------------------------------------------------------------
% timing update
%--------------------------------------------------------------
timingObj=timingObj.timing_update(cycleObj.cycle_timing);
```

This calls the method timing_update, implemented in timingObj. It transfers the current cycle timing to the timing object, for later storage. The input is the cycle timing structure, and the output is the updated timing object.

```
end  % run_cycle
```

This is the end of the cycle.

### 4.3.9 FAR Engine

For a particular example, the FAR engine first defines specific implementations of each of the objects, then runs the perception-action cycle, and completes any storage or display management functions.

### 4.3.10 Utility Functions

The utility functions object contains common utility functions. It consists of:

- Methods: hamming, compute_csigma_ellipse, TimeNowInSeconds.
4.4 The CREW Sensor

The CREW is the world’s first radar test bed built specifically to allow testing of fully adaptive and cognitive algorithms. The CREW was built using an approximately $600K grant from the 2013 DURIP, sponsored by AFOSR. The CREW was designed and specified by CSL director Dr. Graeme Smith and developed by Keysight Technologies (formerly Agilent) and Millitech. The system is a four-channel multistatic radar operating in W-band. There are four pairs of transmit and receive heads allowing full distributed MIMO operation. A block diagram is provided in Figure 4.

The CREW has a fully digital back end, shown to the left in Figure 4, comprising a control PC, four analog-to-digital converters (ADCs) and four arbitrary waveform generators (AWGs). The ADCs and AWGs are connected to the PCI via a PXIe extension system, meaning that the PC is genuinely “in-the-loop” since the transfer rates are high enough that the digitized signals on all four channels can be evaluated and modifications made to the transmit waveforms in real-time. The system can be programmed using MATLAB allowing for rapid prototyping of fully adaptive and cognitive algorithms.

For the radio frequency (RF) front end, shown to the right in Figure 4, the instantaneous bandwidth of the system is 1 GHz, the transmit center frequency is 94 GHz (W-band), the effective radiated isotropic power is 50 dBW, the receiver gain is 55 dB and the receiver noise figure is ≈5 dB. The system is fully coherent across all four channels with a phase stability/error better than 1° root mean square (RMS). A two stage up/downconversion scheme is used and variation in the second local oscillator frequency allows stepped frequency processing across a 4 GHz operational bandwidth.

```
Figure 4: CREW Schematic Showing the Digital Backend, IF Stage, and W-band RF Stage
```

Approved for public release; distribution is unlimited
The CREW is a unique test bed able to facilitate research into fully adaptive and cognitive algorithms and distributed/multistatic operation all within the laboratory. The extremely high bandwidths result in a range resolution of a few centimeters making it possible to observe multiple scattering centers in smaller targets and easy to range-gate out the walls of the laboratory. The W-band center frequency means the narrowband assumption can be made even if the full 4 GHz bandwidth is used. As such it is easy to set-up sophisticated experiments using the CREW within CSL.

However, developing and debugging FAR algorithms while running experiments can be cumbersome and time-consuming. While developing algorithms under the FAR Phase II SBIR, it became apparent that a simulation of the CREW was needed for algorithm development and testing under controlled and reproducible conditions, with the ability to switch between simulated and experimental data sources easily. In this project, we have developed this capability.

The implementation of the CREW Sensor object consists of:

- **Properties**: \( c \), transmitter (structure), receiver (structure), waveform (structure), Simulated, ConfigMaster, ConfigSlave, driver
- **Methods**: Sensor_CREW (constructor), get_measurement, constructWaveform, sendWaveform, receiveDataSim, receiveDataCREW

The property \( c \) is the speed of light and transmitter, receiver, and waveform are structures that contain parameters that characterize the CREW sensor. The parameters we can adapt are in waveform. For the five-parameter example, the parameters are PRF, number of pulses, pulse length, bandwidth, and transmitted power. The flag \( \text{Simulated} \) is set true for simulation and false for experiment. The parameters ConfigMaster, ConfigSlave, and driver are used to interface to the CREW.

The object Sensor_CREW implements the get_measurement method as shown in Figure 5. It contains a switch to toggle between simulated and CREW data. The CREW API consists of the methods sendWaveform and receiveDataCREW. These interface directly with the CREW to transmit pulses and receive echo returns. The simulation API and sensor consists of the methods constructWaveform and receiveDataSim. The method receiveDataSim gets target, clutter and noise parameters from Scene_CREW, and generates random samples of complex clutter data.
Figure 5: CREW Sensor Object
4.5 Summary

This chapter provided a description of the MATLAB-based FAR M&S architecture. The architecture is coded in MATLAB using an OOP approach. It includes a FAR engine to control the operation of the perception-action cycle and software objects that determine the next set of sensing parameters; obtain data from the sensor; process the data to track the target; and store and display the results of the sensing and tracking processes. We have developed modules that implement simulated DSRA and pre-recorded SDR data examples in [30]-[35], and the real-time CREW data examples in [36]-[39]. We have developed a simulation of the CREW and the API layers for the simulated and experimental CREW data sources to enable switching between simulated and experimental data.
5 CONCLUSIONS

The potential of cognitive approaches to enhance existing radar performance in almost all respects has led to an upsurge in research in recent years and a key gap in the Air Force’s radar M&S tools is the lack of a comprehensive, dynamic distributed radar scenario generation capability for distributed FAR systems.

In this project we have developed a MATLAB-based M&S architecture for distributed FAR radar that will enable algorithm development and testing on simulated, previously collected, and real-time streaming data. The architecture is coded in MATLAB using an OOP approach and implements the FAR framework developed in [30]-[35]. It includes a FAR engine to control the operation of the perception-action cycle and software objects that determine the next set of sensing parameters; obtain data from the sensor; process the data to track the target; and store and display the results of the sensing and tracking processes. We have developed modules that implement simulated DSRA example in [33], the pre-recorded SDR data example in [35], and the real-time CREW data examples in [36]-[40]. We have developed a simulation of the CREW and the API layers for the simulated and experimental CREW data sources to enable switching between simulated and experimental data. A demonstration was given in March 2016 for members of the NATO SET-227 Panel on Cognitive Radar.

The FAR M&S architecture developed in Phase I allows for transparent switching between the simulated and experimental CREW data sources, as well as between FAR algorithms that drive the sensing. The ability to easily interchange sensing and processing objects will allow for rapid development and testing of cognitive radar algorithms by structuring the M&S functions to avoid duplicating effort and “single point” solutions. It will enable collaboration between researchers in industry, academia, and the Air Force, as algorithms developed by different researchers can be tested and compared using consistent simulations, collected data, and laboratory conditions. In Phase II, we plan to make the FAR M&S architecture code available to members of the NATO SET-227 Panel on Cognitive Radar. Collaborations with members of this panel to develop and test algorithms on the CREW are already underway. Furthermore, several members have already begun development of their own cognitive radar test beds and our FAR M&S architecture will enable further collaboration within the panel using these data sources.

In Phase II, we also plan extend the baseline architecture to model a dynamic, distributed airborne MIMO radar FAR system using the full MIMO-CMS tool as the simulation base. This will provide a comprehensive radar scenario generation capability that will fill a key gap in the Air Force’s previously developed radar M&S tools.
REFERENCES


# LIST OF ACRONYMS, ABBREVIATIONS, AND SYMBOLS

<table>
<thead>
<tr>
<th>ACRONYM</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2/AD</td>
<td>anti-access/area denial</td>
</tr>
<tr>
<td>ADC</td>
<td>analog-to-digital converter</td>
</tr>
<tr>
<td>AFOSR</td>
<td>Air Force Office of Scientific Research</td>
</tr>
<tr>
<td>AFRL</td>
<td>Air Force Research Laboratory</td>
</tr>
<tr>
<td>API</td>
<td>application programming interface</td>
</tr>
<tr>
<td>AWG</td>
<td>arbitrary waveform generator</td>
</tr>
<tr>
<td>BCRLB</td>
<td>Bayesian Cramér-Rao lower bound</td>
</tr>
<tr>
<td>BIM</td>
<td>Bayesian information matrix</td>
</tr>
<tr>
<td>BLR</td>
<td>Bayesian likelihood ratio</td>
</tr>
<tr>
<td>BLRT</td>
<td>Bayesian likelihood ratio test</td>
</tr>
<tr>
<td>CPI</td>
<td>coherent processing interval</td>
</tr>
<tr>
<td>CR</td>
<td>Cognitive radar</td>
</tr>
<tr>
<td>CREW</td>
<td>Cognitive Radar Engineering Workspace</td>
</tr>
<tr>
<td>CSL</td>
<td>Cognitive Sensing Laboratory</td>
</tr>
<tr>
<td>DSRA</td>
<td>distributed sensor resource allocation</td>
</tr>
<tr>
<td>DURIP</td>
<td>Defense University Research Instrumentation Program</td>
</tr>
<tr>
<td>EFIM</td>
<td>expected Fisher information matrix</td>
</tr>
<tr>
<td>ESL</td>
<td>ElectroScience Laboratory</td>
</tr>
<tr>
<td>FAR</td>
<td>fully adaptive radar</td>
</tr>
<tr>
<td>FFR</td>
<td>feed-forward radar</td>
</tr>
<tr>
<td>FIM</td>
<td>Fisher information matrix</td>
</tr>
<tr>
<td>GMTI</td>
<td>ground moving target indicator</td>
</tr>
<tr>
<td>IF</td>
<td>intermediate frequency</td>
</tr>
<tr>
<td>ILR</td>
<td>integrated likelihood ratio</td>
</tr>
<tr>
<td>LRDT</td>
<td>likelihood ratio detection and tracking</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>modeling and simulation</td>
</tr>
<tr>
<td>MAP</td>
<td>maximum a posteriori</td>
</tr>
<tr>
<td>MAP-PF</td>
<td>maximum a posteriori penalty function</td>
</tr>
<tr>
<td>MBET</td>
<td>monostatic-bistatic equivalence theorem</td>
</tr>
<tr>
<td>MIMO</td>
<td>multiple input multiple output</td>
</tr>
<tr>
<td>MIMO-CMS</td>
<td>MIMO radar clutter modeling and simulation</td>
</tr>
<tr>
<td>ML</td>
<td>maximum likelihood</td>
</tr>
<tr>
<td>MMSE</td>
<td>minimum mean square error</td>
</tr>
<tr>
<td>MSE</td>
<td>mean square error</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>NATO</td>
<td>North Atlantic Treaty Organization</td>
</tr>
<tr>
<td>OOP</td>
<td>object oriented programming</td>
</tr>
<tr>
<td>OSU</td>
<td>The Ohio State University</td>
</tr>
<tr>
<td>PA</td>
<td>perception-action</td>
</tr>
<tr>
<td>PDF</td>
<td>probability density function</td>
</tr>
<tr>
<td>PC-BIM</td>
<td>predicted conditional Bayesian information matrix</td>
</tr>
<tr>
<td>PC-CRLB</td>
<td>predicted conditional Cramér-Rao lower bound</td>
</tr>
<tr>
<td>PIM</td>
<td>predicted information matrix</td>
</tr>
<tr>
<td>PRF</td>
<td>pulse repetition frequency</td>
</tr>
<tr>
<td>RCS</td>
<td>radar cross section</td>
</tr>
<tr>
<td>RF</td>
<td>radio frequency</td>
</tr>
<tr>
<td>RLSTAP</td>
<td>Research Laboratory Space Time Adaptive Processing</td>
</tr>
<tr>
<td>RMS</td>
<td>root mean square</td>
</tr>
<tr>
<td>SDR</td>
<td>software defined radar</td>
</tr>
<tr>
<td>SET</td>
<td>Sensors Electronics Technology</td>
</tr>
<tr>
<td>SIMO</td>
<td>single input multiple output</td>
</tr>
<tr>
<td>SISO</td>
<td>single input single output</td>
</tr>
<tr>
<td>SMS-MBS</td>
<td>Signal Modeling and Simulation Tool for Multichannel Bistatic Systems</td>
</tr>
<tr>
<td>SNR</td>
<td>signal-to-noise ratio</td>
</tr>
<tr>
<td>SPC</td>
<td>Signal Processing Consultants, Inc.</td>
</tr>
<tr>
<td>STAP</td>
<td>space-time adaptive processing</td>
</tr>
</tbody>
</table>