IEEE ComSoc IEEE Communications Society | IEEE COMPUTER SOCIETY | IEEE Signal Processing Society | VTS | IEEE Transactions on Machine Learning in Communications and Networking

Received 7 February 2024; revised 29 May 2024; accepted 21 August 2024.
Date of publication 26 August 2024; date of current version 2 September 2024.

The associate editor coordinating the review of this article and approving it for publication was G. Yu.

Digital Object Identifier 10.1109/TMLCN.2024.3449834

ML-Enabled Millimeter-Wave Software-Defined Radio With Programmable Directionality

MARC JEAN, MURAT YUKSEL (Senior Member, IEEE), AND XUN GONG (Senior Member, IEEE)

Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816 USA

CORRESPONDING AUTHOR: M. JEAN (ma935977@ucf.edu)

This work was supported in part by U.S. National Science Foundation under Award 1836741.

ABSTRACT The increasing demand for gigabit-per-second speeds and higher wireless node density is driving the need for spatial reuse and the utilization of higher frequencies above the legacy sub-6 GHz bands. Since these super-6 GHz bands experience high path loss, directional beamforming has been the main method of access to the large amount of bandwidth available at these higher frequencies. Hence, the programming of wireless beams with specific directions is emerging as a requirement for software-defined radio (SDR) platforms. To address this need, we introduce an affordable millimeter-wave (mmWave) testbed. Using a multi-threaded software architecture, the testbed allows for the convenient programming of mmWave beam directions using a high-level programming language, while also providing access to machine learning (ML) libraries as well as SDR methods traditionally deployed in Universal Software Radio Peripheral (USRP) devices. To showcase the potential of the testbed, we tackle the Angle-of-Arrival (AoA) detection problem using reinforcement learning (RL) methods on the receiver side. AoA detection and direction finding is a crucial need for the emerging use of super-6 GHz spectra. We design and implement Q-learning, Double Q-learning, and Deep Q-learning algorithms that passively inspect the Received Signal Strength (RSS) of the mmWave beam and autonomously determine the predicted AoA. The results indicate the feasibility of programming directionality of the wireless beams via ML-based methods as well as solving difficult problems pertaining to emerging directional wireless systems.

INDEX TERMS Beamforming, SDR, mmWave, USRP, ML, Q-learning, double Q-learning, deep Q-learning, AoA.

I. INTRODUCTION

SINCE legacy sub-6 GHz bands are highly contested, the wireless community has turned to super-6 GHz spectra for addressing the capacity problem [3], [4], [5] in the emerging 5G [6] and the envisioned 6G [7] standards. Yet, accessing the super-6 GHz bands requires directionality in wireless signal propagation. Due to strong attenuation at these frequencies, directional beams are employed to reach longer ranges with the same transmit power. Directional mmWave [8], [9] approaches are being tried for higher capacity at the expense of less mobility, as these bands are fragile to mobile operation and experience high path loss in non-line-of-sight (NLoS) channels. In order to successfully integrate these super-6 GHz bands (22 GHz and above including the Terahertz and free-space optical bands) into legacy mobile networking at scale, there is a need for joint hardware and software innovations.

Directionality offers wideband, energy-efficient, and inherently secure wireless communications. While offering higher spatial reuse and less interference [10], [11], [12], directional transceivers have key advantages in (+) wide bandwidth and effective data rate as they operate with much shorter wavelengths than legacy sub-6 GHz bands, (+) energy efficiency (i.e., low energy-per-bit transfers) as they dissipate transmit power to a smaller volume, and (+) security (i.e., low probability-of-intercept) as they attain better

containment of the radio signal and enable innovative spatial security solutions such as null space beamforming [13]. On the other hand, directionality has disadvantages in terms of tolerance to mobility and antenna size. Higher directionality requires (-) transmitter and receiver to be facing towards each other, i.e., line-of-sight (LoS) establishment or alignment, and (-) larger antenna form factor to realize high gain which forces directional transceivers to be plausible only at high frequency bands with significant path losses.

Due to their directionality, attaining mobility at super-6 GHz frequencies is challenging mainly because directional beams require constant attention for maintaining LoS alignment. Beamsteering antennas integrated with SDRs offer an opportunity to solve these challenges arising from directionality in mobile settings. Very interesting set of 'software-defined beamforming' designs become possible, if such integration of beamsteering antennas with SDRs is done at scale and in a manner that enables advanced algorithmic methods in beamforming technologies. Equipping legacy SDR platforms with an application programming interface (API), such as `setDirection()` or `setAngle()` functions to manipulate directionality, enables ML-based treatment of beam directions as well as PHY-MAC hybrid designs involving directional beamforming while maintaining multiple simultaneous links with neighbors. Unlike traditional tracking and acquisition done in hardware, such *Directional SDR* enables handling of LoS detection and establishment via software.

Realizing *programmable directionality* as a prominent approach in SDRs necessitates seamlessly coupling high-level programming environments (where advanced algorithmic methods such as ML and data structures are available) and directional beamforming capability in the PHY layer. In this work, we present an mmWave SDR testbed that enables ML-based programming of the directional features of the beams while keeping the data-plane processing over field-programmable gate array (FPGA). The testbed employs a USRP device and conveniently allows Python programming of directionality of the mmWave beams. We use the testbed in mmWave AoA detection and demonstrate the application of RL methods in real time.

Direction finding is a crucial need for the emerging highly directional mmWave systems. The detection of AoA for RF signals, as discussed in [14], holds a pivotal role in enhancing the alignment of mmWave beams. This capability is a crucial aspect of directional wireless technology, employed for identifying signals transmitted within the surrounding environment, as emphasized in [15]. Precise AoA estimation facilitates the fine adjustment of beam alignment between transmit and receive antennas, resulting in more accurate Channel State Information (CSI). Consequently, this leads to an enhancement in RSS, contributing to an improved overall signal-to-noise ratio (SNR) and enhanced link performance. A significant portion of the available literature focuses on the utilization of phase or complex signals to determine the radio source's direction [16]. Employing either of these approaches for pinpointing the signal source's direction entails repetitive sampling. Consequently, the algorithm may necessitate a significant time investment to detect the signal source, rendering it less suitable for real-world applications, especially in scenarios involving fast-moving targets.

For mmWave AoA detection, we adapt model-free RL algorithms. Our RL-based approach to detecting AoA is compatible with mmWave SDR systems as we show it in our testbed. Our approach only considers the receiver side and can passively detect AoA without help from the transmitter or any other localization system. We utilize three RL methods, Q-learning, Double Q-learning, and Deep Q-learning for AoA detection. Q-based learning algorithms are widely used for a wide variety of applications that require fast learning capability, such as in gaming, or fast detection capability, such as detecting a drone flying through an indoor environment [17]. Our RL algorithms for AoA detection follow a Markovian model, using actions to explore different states of a given environment [18], [19]. We design a reward function that does not necessitate anything other than passive measurements of RSS and tune the algorithms' greediness and learning rates to obtain a balance between faster convergence and accuracy in AoA detection.

Our work aims to address experimental challenges in integration of hardware and software components for mmWave systems. A key goal is to demonstrate the feasibility of using machine learning (ML) in solving various challenges of mmWave systems due to their directional operation. As a key application, we focus on AoA detection and direction finding using RSS. Main contributions of our work are as follows:

- Design and implementation of a low-cost SDR-compatible testbed platform for ML-in-the-loop mmWave beamforming.
- Design of a multi-threaded software integration method that enables handling of timescale difference and asynchrony between a high-level programming environment, Python, and the underlying mmWave (or super-6 GHz) antenna system.
- A modular API that can be adapted for a variety of mmWave antennas, such as, horn or patch antenna arrays.
- Adaptation of Q-learning, Double Q-learning, and Deep Q-learning methods for mmWave AoA detection.
- Tuning the hyper parameters (the learning rate α and exploration policy ϵ) of both Q- and Double Q-learning to detect AoA within 2° of accuracy.
- Tuning the hyper parameter exploration policy ϵ and the number of hidden layers of Deep Q-learning to detect AoA within 2°.
- Design of a threshold-based convergence criteria for both Q-learning, Double Q-learning, and Deep Q-learning using coefficient of variation (CoV) of RSS data samples.

The rest of the paper is organized as follows: Section II covers the related literature on experimental mmWave SDR efforts and AoA detection. Section III presents the

architecture design of our experimental platform and how AoA detection algorithms may be implemented on it. Section IV presents our RL algorithms for AoA detection and discusses results from their implementation on the testbed. Finally, Section V makes a summary of our work and discusses future directions.

II. BACKGROUND AND RELATED WORK

In recent years, there has been an exponential growth in mobile users [20]. This has caused a significant increase in network traffic. With the deployment of highly dense Internet-of-Things (IoT) networks, the demand for larger data rates is increasing. To address this issue, upcoming 5G networks are tapping into unlicensed mmWave bands, beyond 22 GHz [21]. Such super-6 GHz bands enable data rates on the order of gigabits per second (Gbps). While using mmWave bands can have numerous advantages, the short wavelength is impacted by environmental factors, primarily absorption [22]. mmWave signals are also fragile in mobile applications, due to multi-path propagation caused by environmental reflection. As a result of these environmental losses, mmWave signals can severely degrade with respect to distance traveled.

To combat environmental loss, high gain mmWave antennas with beamforming features are used to aim the radiated power to desired directions. Using highly directional antennas have numerous beneficial features. If the beam is aligned well, the RSS is notably better than omni-directional signals, yielding a better SNR. As a result, the user data rates will increase due to the directional beams offered by the mmWave beamforming antennas. Further, directionality offers energy efficient communication links, enabling the spatial reuse of frequencies that are available in the spectrum. However, directionality is difficult to work with under mobility as it brings the problems of frequent beam alignment and maintenance of LoS.

The mmWave antenna beam direction can be steered mechanically or electronically. Older mmWave platforms mechanically steered horn antennas to align the transmit and receive beams [23]. Today, more advanced patch antenna array systems are used to electronically steer the beams. Usually, digital phase shifters are used to control the beam direction via software. Beamforming, while optimizing for several desired metrics (e.g., throughput, security, and energy), using these antenna systems proved to be difficult – triggering the need for advanced algorithmic methods, such as ML, in SDR platforms [24], [25], [26]. Integrating these highly directional beamforming mmWave antennas into general purpose networking platforms is an open research problem [27]. This requires both software and hardware solutions enabling mmWave experimentation capability with ML methods while being able to work with varying mmWave antenna hardware in a cost-efficient manner [28], [29]. Our work fills this gap by integrating a multi-threaded software design with a conventional SDR in a testbed, and showcasing the practicality of the approach on solving the problem of AoA detection by applying RL algorithms. Hence, our work relates to mmWave testbed design as well as AoA detection, both of which we survey next.

A. mmWAVE TESTBEDS

Most of the software-controlled mmWave experimentation has utilized FPGAs to handle the wireless signals. OpenMilli [30] utilized an FPGA-based data plane while accessing mmWave bands with a patch antenna array. Following this seminal effort, several studies expanded mmWave experimentation capabilities using FPGA-based designs. A 12-element phased array [31] emulated PHY and enabled studying various aspects of indoor 60 GHz links. Tick [32], offering programmability using XML, expanded the experimentation capability to MAC along with mmWave PHY. By separating control and data paths to different multiple RF chains, M-Cube [33] demonstrated an improved mmWave multi-input multi-output (MIMO) experimentation capability. Being limited to a single datastream (i.e., communication can take place only with one neighbor node at a time), the design included a USRP that guides an FPGA on the control path while a separate data path connects to the host machine. MilimeTerra [34] envisioned a similar capability with commercial off-the-shelf mmWave or Terahertz antennas.

There have also been experimental efforts to investigate mobility effects on mmWave systems. As we discussed earlier, dynamic channel conditions generated by the mobility necessitates software techniques to manipulate the directions of the mmWave beams. In [35], the authors, using FPGA to process the RF signals, showed software implementation of a beam alignment method for IEEE 802.11ad. A mobile Orthogonal Frequency-Division Multiplexing link was demonstrated [36] using a 64-element mmWave phased array where the baseband module was switched between an FPGA and USRP to overcome USRP's limited bandwidth. The FPGA in the baseband module controls the phased array's weights, which limits the programmability.

A drawback of the FPGA-based designs is that their programmability is limited to the hardware language. Even though FPGAs do act as programmable hardware interfaces, translation of high-level programming languages to hardware languages of FPGAs (e.g., Verilog or VHDL) is limited to certain commands as FPGAs have to be pre-configured before operation. This disallows the capability of programming directionality (of potentially multiple antennas) in real time. SDR designs using USRPs can utilize a large swathe of ML and multi-threaded algorithmic methods available at high-level languages. The closest to our work was [37], where a USRP-based mmWave testbed, utilizing 60 GHz horn antennas, was shown to capture channel measurements using GNU Radio. The main difference of our testbed is the beamsteering capability controlled from a high-level programming language, i.e., Python. This capability allows us to define programmable directional software interfaces, which

can be used to implement a variety of algorithms that can steer directions of the mmWave beams while performing other tasks, such as sensing and computing, that run in parallel using other threads. This approach allows the SDR programmer to conveniently utilize directionality of the beams as part of sophisticated software methods to attain higher level goals, e.g., beam alignment [38], beam discovery and tracking [39], or AoA detection [40].

B. AoA DETECTION

AoA detection or direction estimation/finding has been an extensively studied problem within the context of wireless localization [41]. With the recent advent of directional beamforming capabilities in super-6 GHz systems, AoA detection, in particular, has gained a renewed interest due to emerging applications using such systems [42]. Researchers have relied on virtual environments and simulations to perform AoA detection or directionality in mmWave bands. These virtual environments have gotten more sophisticated with the usage of 3D ray tracing. In [43], 3D ray tracing is used to simulate mmWave signals in virtual environments. Users can use the open source software to design large intelligent reflective surfaces and determine AoA using compressive sensing algorithms. Although using simulation-based approaches is cost effective, they do not render the physical world and fall short of precisely modeling complicated physical communication channel dynamics in mmWave or other super-6 GHz bands.

Recently, cheaper off-the-shelf SDRs have been used to setup testbed platforms for AoA detection. The testbed platform [15] uses a Kerberos radio with four whip antennas at the receiving end. At the transmitting end a long range (LoRa) radio is used to transmit a signal at 826 MHz. LoRa is beneficial for long range communication and uses low transmit power. The transmitter includes a GPS and compass unit used to label the direction of the transmitted signal. The labeled data set is the ground truth that is trained in the ML algorithm. The data is trained using a deep learning Convolutional Neural Network (CNN) model [15].

Multiple Signal Classification (MUSIC) is a widely used AoA detection algorithm and assumes that the received signal is orthogonal to the noise signal. MUSIC uses this assumption to decompose the noise from the received signal into separate sub-spaces. The power spectral density (PSD) of the signal is taken as a function of angle [44]. The angular value which results in the maximum power is estimated to be the AoA. The assumption that the received signal is orthogonal to the noise signal is not valid in real world scenarios. Therefore, MUSIC does poorly in environments that involve NLoS propagation. Since mmWave signals can experience severe environmental path loss and involve multiple NLoS signals, MUSIC may not be a good choice for mmWave AoA detection.

Support Vector Regression (SVR), a supervised ML algorithm, has also been used to estimate AoA. Regression

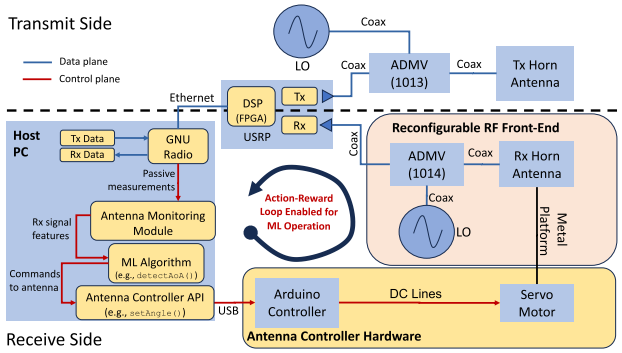


FIGURE 1. Testbed architecture.

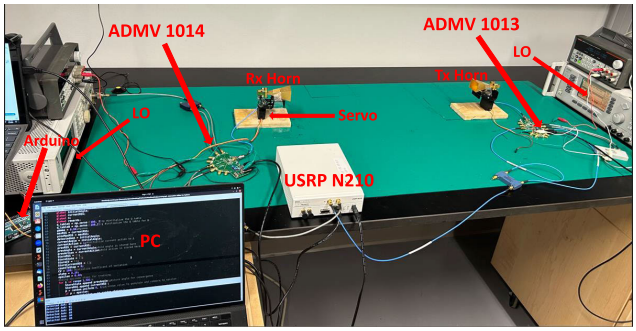


FIGURE 2. Testbed configuration.

does poorly in estimating AoA from impinging signals at multiple time steps [15]. The algorithm cannot be used to determine AoA since the number of impinging signals is unknown [45]. As a result, the algorithm can be used for detecting AoA for a single source at a time. This makes SVR less robust for AoA detection in environments with multiple signal sources. Therefore, SVR is not a good choice for mmWave AoA detection. The CNN model used in [15] adapts a hybrid configuration. A classification method is used to determine the number of impinging receive signals and two regressive heads are used to determine the AoA. The study showed that CNN outperformed the other classical ML methods, MUSIC and SVR. Further, the CNN model was able to estimate AoA within 2° of error.

To the best of our knowledge, our RL-based AoA detection approach is the first to apply RL on RSS measurements. Previous efforts used either (i) supervised learning on the RSS (or other signal) measurements [15], [45] or (ii) multi-element receiver antenna to detect phase differences [44]. The former methods require training of the machine learning algorithm and the latter requires multiple antennas on the receive side and coherency in the transmitted signal's phase that is orthogonal to noise. The key difference of our AoA detection approach is that it does not use supervised learning. Our RL-based AoA detection methods do not require the ground truth and determine the AoA based only on the RSS observations at the receiver.

III. mmWAVE TESTBED ARCHITECTURE AND DESIGN

The testbed design is centered around three goals: (1) enabling the programmability of mmWave beam's direction from a high-level programming language, (2) enabling access to existing (and future) advanced algorithmic techniques in cognitive radio literature, and (3) using open-source modular software to provide a high degree of programmability in communication components. For the first, second, and third goals, we respectively use Python code (some of which is produced by GNU Radio), USRP, and GNU Radio. Figure 1 shows a block diagram representation of the overall testbed architecture. The architecture utilizes a multi-threaded software integration (in the PC in Figure 1) to establish a closed-loop operation capability. This is needed for ML operation to implement an action-reward loop, i.e., the ML algorithm takes an action by updating the mmWave antenna's beamforming configuration and that results in a change on the reward which is observed by inspecting the received data signal. Due to the timescale difference, each software module in the host PC is a separate thread coordinated via a timer.

We focused on showing the proof-of-concept and opted for the most inexpensive way of building the testbed. The effective bandwidth of the testbed can be improved with higher end components by using the same architecture. A USRP N210 is used to transmit and receive signals via coaxial connections through the RF frontends and antennas. Figure 2 presents a picture of the actual testbed configuration with labels mentioned in the block diagram.

A. INTEGRATION WITH USRP

1) RF CHAIN

External RF frontends are used to up/down convert the transmit and receive signals to address the frequency limitation of USRP devices. Shown in [46], RF mixers and amplifiers are connected in series to up/down convert the signals. This approach is costly, due to the high cost of the individual components. Further, connecting numerous devices adds weight and makes the testbed less mobile. To up/down-convert the transmit and receive signals we use off-the-shelf mmWave frontends, ADMV1013 [47] and ADMV1014 [48]. These RF frontends can operate at 24-44 GHz and are manufactured on PCB boards making them low in cost. Signal generators connected to the RF frontends are used to produce the local oscillator (LO) signals used for mixing.

2) ANTENNAS

We used two K band horn antennas that operate at 18-27 GHz. The antennas have a gain of 15 dBi. On the Rx side, the antenna is mounted onto an MG995 servo that can be steered by an Arduino micro-controller which is connected to a PC via USB. The servo is powered by the Arduino's 5V DC port. A pulse width modulated (PWM) signal, generated by the Arduino, can rotate the servo within [0°,180°]. Instead of horn antennas, electronically steered antenna arrays, such as phased arrays, can be integrated to the testbed by controlling the bias voltages of the digital phase shifters via the PWM signals from the Arduino. A phased array antenna is currently under development, and the use of horn antennas does not compromise the testbed's architectural value.

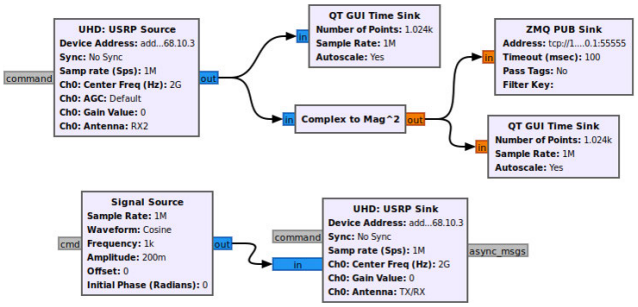


FIGURE 3. GNU radio software configuration.

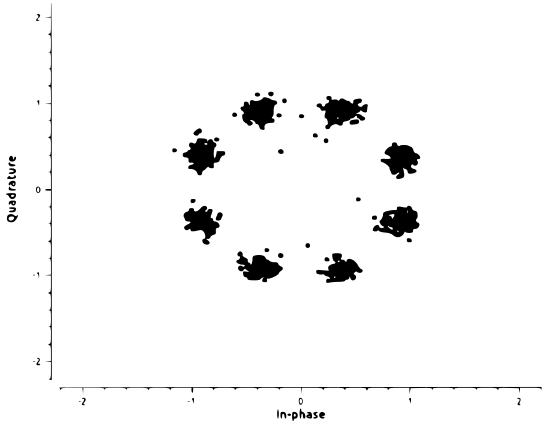


FIGURE 4. 8-PSK demodulation.

3) mmWAVE WITH GNU RADIO

GNU Radio is open source and can be used with high-level programming languages, such as, Python. This makes it convenient to work with the USRP. It has built-in digital communication and signal processing blocks, such as modulation schemes, filters, and Fast Fourier Transform, available via a Graphical User Interface (GUI). These GUI blocks are implemented in Python and the user does not have to program them from scratch. As a result, GNU radio has been widely used, by the cognitive radio community. Figure 3 presents a simple GNU Radio GUI block interface used to transmit and receive a Cosine signal over the USRP. The UHD:USRP Sink block is used to tune and transmit the signal from the device. The UHD: USRP Source block is used to process the receive signal and output complex base-band samples. The two blocks communicate with the USRP N210 via an Ethernet connection.

The USRP N210's maximum sampling frequency is 25 MHz via 1 Gigabit Ethernet connection. In the setup in Figure 3, the sampling rate is set to 2.5 MHz. The Tx and Rx signals are tuned to 2 GHz center frequency. Since we are working with a mmWave channel, the Tx signal is up-converted to 26 GHz and down-converted to 2 GHz at

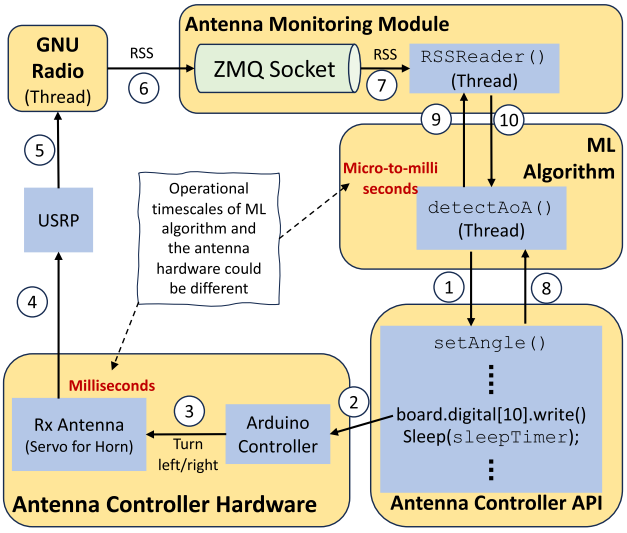


FIGURE 5. Software architecture: Ideal sequence of events for accurate measurement of RSS while there is timescale difference between the ML algorithm and antenna hardware.

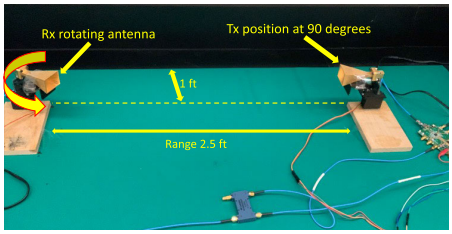


FIGURE 6. Aligned: Transmitter at 90°.

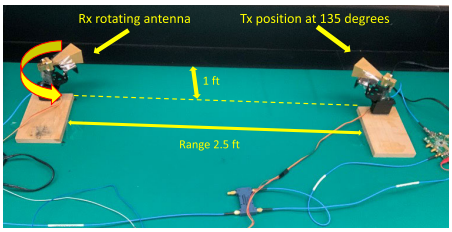


FIGURE 7. Misaligned: Transmitter at 135°.

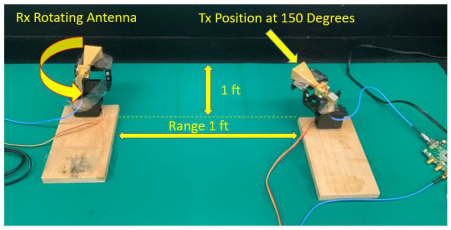


FIGURE 8. Misaligned: Transmitter at 150°.

reception. Figure 4 is an example of a demodulated 8 phase shift keyed (8-PSK) signal that was transmitted at a carrier of 26 GHz.

B. PROGRAMMING DIRECTIONALITY IN PYTHON

GNU Radio is an excellent platform for manipulating several radio properties from a high-level programming language such as Python. However, software interfaces that allow easy manipulation of beam direction are mostly lacking. Programmer typically has to figure out the best radio configuration for beamforming, mostly in an antenna-specific manner [46], [49], [50], [51]. Software constructs that will allow the programmer to conveniently program the directionality of the mmWave beams while working with other physical layer radio parameters are heavily needed. In this section, we illustrate one such design which uses multiple threads while building directional SDR capabilities.

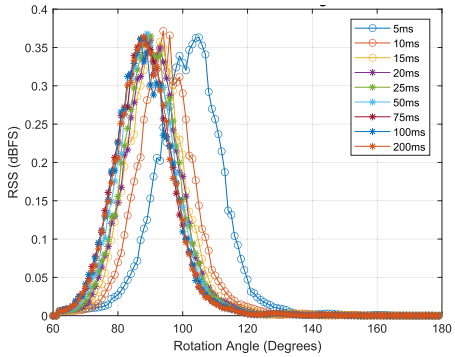


FIGURE 9. Full Sweep: Transmitter at 90°.

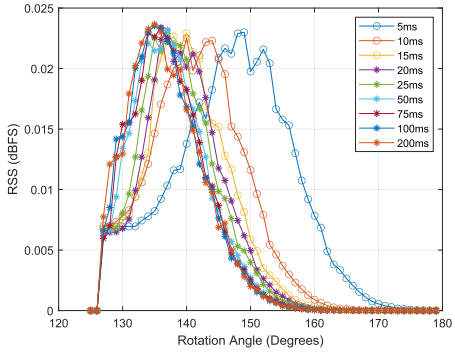


FIGURE 10. Full Sweep: Transmitter at 135°.

The flow of our software design at the receiver side can be seen in Figure 5. The figure shows the ideal sequence of events that needs to take place for an accurate measurement of RSS to be provided to the ML algorithm. The horn antenna can only be steered via the servo, controlled by the Arduino. Along with the antenna steering capability, we aimed to retrieve the received signal strength (RSS) so that ML-based advanced tasks such as AoA detection can be easily programmed. Our ML algorithm is implemented within the `detectAoA()` function, which continuously monitors the change in RSS received from `RSSReader()`. `RSSReader()` receives RSS data via a ZMQ socket connection from GNU Radio. The connection and `RSSReader()` implement the Antenna Monitoring Module and can be reconfigured to receive other RF signal parameters such as phase and frequency. The ML algorithm for `detectAoA()` uses RSS data in calculating its reward value so as to decide which action to take.

When an action is taken by the ML algorithm, it needs to be translated into beamforming steps for the antenna. In our design, the ML algorithm thread makes a *blocking call* to the Antenna Controller API to send the beamforming command

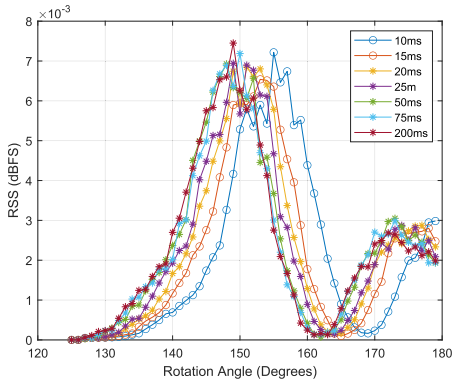


FIGURE 11. Full Sweep: Transmitter at 135°.

to the Antenna Controller Hardware. This design makes it modular to customize the Antenna Controller API without having to change the antenna hardware. The `setAngle()` function implements the API by executing the command to rotate the servo to the desired angle. After this command (which executes the beamforming action), `setAngle()` executes a sleep for a duration of `sleepTimer` to make sure its caller thread, i.e., `detectAoA()` waits before calling `RSSReader()` to observe the result of its action. This design segregates the antenna's execution of the action from the API's implementation, which enables modularity of the API. An alternative to this approach would be to make the API's execution of the beamforming action command (i.e., board.digital[10].write() for the servo-based horn antenna) a blocking call, which would make the thread executing `setAngle()` (and hence its caller `detectAoA()` wait until the antenna completes the beamforming action. However, not all antennas can accommodate this as it requires the antenna to report the status of the command back to the API. Thus, we chose to make the command non-blocking.

The multi-threaded design with blocking and non-blocking calls make it necessary to tune timescales of threads running different components of the software system. Figure 5 shows the right sequence of events, (1) – (10), in order for the ML algorithm thread to accurately read the new RSS value after sending a beamforming command to the directional antenna control API. Further details of the software design of the testbed, including code snippets, are available in [1].

C. TUNING `sleepTimer`: A NAIVE AoA DETECTION ALGORITHM

A key parameter to tune for the testbed is `sleepTimer` as setting it too small can cause inaccurate measurements and too large will slow down the testbed operation unnecesarily. To observe `sleepTimer`'s effect on the performance and tune it properly, we implemented a naive AoA detection algorithm and measured the accuracy of the AoA detection as `sleepTimer` varies. The algorithm simply scans from the Rx antenna's starting angular position to 180° and predicts AoA to be the angle yielding the highest RSS. We tested the AoA detection algorithm in three different scenarios, shown in Figs 6-8. The first scenario (Figure 6) has the Tx antenna in perfect alignment with the Rx antenna at 90°. The Rx antenna starts from 60° and scans up to 180°. The second (Figure 7) and third (Figure 8) scenarios involve misalignment, where Tx antenna is positioned at 135° and 125°, respectively, towards the wall. In these two scenarios, the Rx antenna starts scanning from 125° up to 180°. In the first two scenarios, the Tx antenna is a K band Horn antenna sending a 26 GHz signal, and the Tx and Rx antennas are 2.5ft apart and placed 1ft away from the wall. The Tx antenna in the third scenario is a Ka band Horn antenna transmitting at 30 GHz. Another difference of the third scenario is that the antennas are closer to each other at 1ft apart. In all the scenarios, we measured the RSS and AoA by using different `sleepTimer`, from 5ms to 200ms, in the `setAngle()` function.

The impact of `sleepTimer` on the accuracy of RSS measurements and the AoA detection is observable when it is comparable to the amount of time it takes to steer the Rx antenna. As shown in Figure 9 of the first scenario, the RSS increases as the Rx antenna is rotated in increments of 1°. As `sleepTimer` becomes faster, the measured RSS becomes less accurate with a lag across the rotation angle. This is because the thread measuring the RSS does not wait long enough for the Rx antenna to get to its correct orientation after receiving the command to steer 1°. This inaccuracy becomes unacceptable when `sleepTimer` is lower than 15ms, causing the peak RSS to occur at a rotation angle notably larger than desired AoA of 90°'s. When there is a misalignment, AoA detection becomes more complicated due to multipath signals reflected from the environment. In the misalignment scenarios (Figures 7 and 8), we observe that the impact of faster `sleepTimer` is stronger as seen in Figures 10 and 11.

IV. RL-BASED mmWAVE AoA DETECTION

We designed RL algorithms for AoA detection at the receiver side by only using passive RSS measurements and implemented these algorithms in our testbed. Our aim is to illustrate the use of the testbed for designing a new set of ML-based methods to solve a contemporary problem in super-6 GHz bands. This section presents the RL algorithm designs and their performance.

A. Q-LEARNING AND DOUBLE Q-LEARNING

Q-learning and its variant Double Q-learning are model-free RL algorithms. Both algorithms are based on a Markovian approach that selects random actions [18]. The block diagram in Figure 12 presents our RL model. The learning agent is located at the receiver side with the horn antenna that is mounted onto a servo motor that can be steered. The agent can choose to take two possible actions: move left or right by one degree resolution. Since the servo can rotate up to a maximum angular value of 180 degrees, the time-variant state values can be $s_t \in (0, 180)$. For our setup, a positive reward (i.e., when the action improves the RSS) is set to the current RSS_t. A negative reward (i.e., when the action reduces the

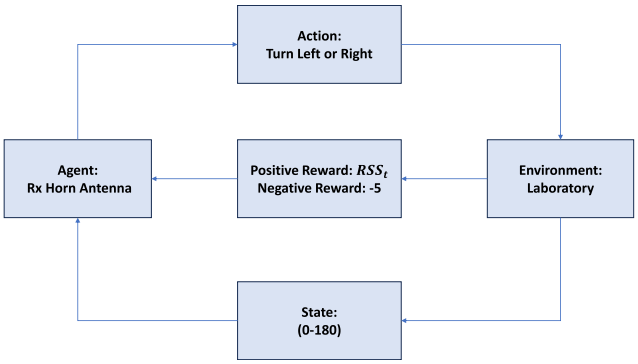


FIGURE 12. **RL configuration.**

RSS) is set to -5. This design incentivizes the agent to seek the angular position that maximizes the RSS, which is implied when the antenna is steered to the correct AoA. Though it is possible to design a more complex state space with more contextual information, we have chosen to use the latest angle only to define the state space. This is a simplifying choice as it assumes zero knowledge of the transmitted beam as well as previous steering angles. If more memory space is available, it is possible to use previous steering angles when defining the state space. Further, if more information about the transmitter characteristics (e.g., the maximum possible signal strength) is available, quantization of RSS becomes possible. This could enable state spaces that are aware of recent RSS measurements (e.g., RSS_{t-1} and RSS_{t-2}). In this study, we choose to use the angle only to define the state space.

By exploring possible states via random actions, Q-learning observes rewards for its actions and stores its learned experience in a Q-table. To populate the Q-table as a function of state, action, and tunable parameters, it uses the following Bellman's Equation [17]:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha * (r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

where s_t is the current state, a_t is the current action, r_t is the reward observed after moving to the current state, s_{t+1} is the next state that was chosen by the agent, $\gamma \in (0, 1)$ is the discount factor, $\epsilon \in (0, 1)$ is the greedy policy, and $\alpha \in (0, 1)$ is the learning rate. Based on the ϵ-greedy policy, the agent chooses either a random action or the state that offers the highest possible reward in the Q-table (i.e., $\max_a Q(s_t, a)$). Smaller values of ϵ cause the algorithm to exploit the Q-table, by searching for the action that results in the largest Q value. Increasing ϵ increases the likelihood that the algorithm will explore the environment, by selecting random actions. γ and α are factors to tune how much the existing Q-table impacts the learning and how fast are the Q-table values updated. For our algorithm, we have set $\gamma = 0.98$ based on earlier studies encouraging high discount factor values [17]. We have set ϵ and α each to 0.1, 0.2, 0.3, and 0.4, generating 16 combinations.

Q-learning stops when RSS samples have a consistent CoV as this means the agent has learned the best AoA and stabilized on it. Algorithm 1 shows the pseudo-code of our

Algorithm 1 AoA Detection With Q-Learning

Input: $\alpha \in (0, 1)$, the learning rate; $\epsilon \in (0, 1)$, the greediness policy; *tickCount* $\in (5, 30)$, the number RSS samples maintained for implementing convergence criterion; *Threshold* $\in (0.01, 1.5)$ threshold for convergence criterion.
Output: *detectedAoA*, AoA detected by the algorithm.
$\gamma \leftarrow 0.98$ ▷ Initialize the discount factor
$RSS[] \leftarrow [\,]$ ▷ Initialize the array of RSS samples
$CoV \leftarrow 1$ ▷ Initialize the coefficient of variation of RSS samples
Randomize *currentAngle* ▷ Initialize to a random angle
$s_t \leftarrow$ *currentAngle*
previousRSS $\leftarrow$ Measure RSS at *currentAngle*
RSS.append(*previousRSS*) ▷ Store RSS sample
sampleCount $\leftarrow 1$
$Q \leftarrow$ Zeros ▷ Initialize the Q-table to 0s
while *Threshold* $\leq$ *CoV* **do**
  RSS.append(RSS_t) ▷ Store RSS data in array
  if Uniform(0,1)$< \epsilon$ **then**
    $a_t \leftarrow$ round(Uniform[0,1])
  else
    $a_t \leftarrow \arg\max_a Q(s_t, a)$
  end if
  if $a_t == 0$ **then**
    currentAngle $--$; ▷ Turn antenna left by 1 degree
  else
    currentAngle $++$; ▷ Turn antenna right by 1 degree
  end if
  newRSS $\leftarrow$ Measure RSS at *currentAngle*
  RSS.append(*newRSS*) ▷ Store the new RSS sample and remove the oldest sample if needed
  sampleCount $++$;
  $s_{t+1} \leftarrow$ *currentAngle*
  $\Delta RSS \leftarrow$ *newRSS* $-$ *previousRSS* ▷ Calculate the reward
  if $0 < \Delta RSS$ **then**
    $r_t \leftarrow$ *newRSS* ▷ Positive reward
  else
    $r_t \leftarrow -5$ ▷ Negative reward
  end if
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha * (r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
  $s_t \leftarrow s_{t+1}$ ▷ Update current state with next state value
  if *tickCount* $\leq$ *sampleCount* **then**
    CoV $\leftarrow$ *RSS.std*()/*RSS.mean*() ▷ Compute the CoV of RSS samples
  end if
end while
return s_t

Q-learning implementation. First, the hyper-parameters γ, α, and ϵ are initialized. The current state s_t is initialized to a random angular value, based off our experimental scenario. For the 90° experimental setup, the random angle can be any value in [80,100] degrees and [120,140] degrees for the 130°. The Q table used to store the Q values from (1) is initialized to zeros. An outer while loop comparing the *Threshold* value and *CoV* is used to decide the stopping condition for the algorithm until the threshold is met. Implementing the ϵ-greedy policy, if the ϵ is greater than a random number in (0,1), the agent takes a random action, i.e., either turn the antenna left or right with one degree resolution. If ϵ is less

than the random value, then the algorithm will exploit the Q table by selecting the action that results in the largest Q value, i.e., the potential reward. A positive reward value is given if ΔRSS is greater than zero. If ΔRSS is less than zero, then the action resulted in a decrease in RSS. Therefore, the action yields a negative reward value of -5.

Algorithm 2 Double Q-Learning

```
    Lines 1-10 in Algorithm 1
11: Q^A ← 0                          ▷ Initialize the first Q-table to 0
12: Q^B ← 0                          ▷ Initialize the second Q-table to 0
13: while threshold <= CoV do
14:     RSS.append(previousRSS)      ▷ Store RSS sample
15:     if Uniform(0,1)< ϵ then
16:         a_t ← round(Uniform[0,1])
17:     else
18:         Q^AB(s_t) ← Q^A(s_t) + Q^B(s_t)
19:         a_t ← arg max_a Q^AB(s_t, a)
20:     end if
21:     Lines 19-35 Algorithm 1
22:     if Uniform(0,1) < 0.5 then
23:         Q^A(s_t, a_t)+ = α * (r_t + γQ^B(s_t+1, arg max_a ×
              Q^A(s_t+1, a)) − Q^A(s_t, a_t))
24:     else
25:         Q^B(s_t, a_t)+ = α * (r_t + γQ^A(s_t+1, arg max_a ×
              Q^B(s_t+1, a)) − Q^B(s_t, a_t))
26:     end if
27:     Lines 37-40 Algorithm 1
28: end while
29: return s_t
```

Q-learning uses a single estimator $\max_a Q(s_{t+1}, a)$, i.e., the next state s_{t+1} that has the highest Q value among all possible actions. As shown in [19], this causes the algorithm to overestimate the desired solution, by causing a positive bias. As a result, Q-learning can perform poorly in certain stochastic environments. To improve the performance of Q-learning, other variants, such as Double Q-learning [19], Delayed Q-learning [52], Fitted Q-iteration [53], and Phased Q-learning [54], were developed to improve accuracy and convergence time. To reduce overestimation, the Double Q-learning variant uses two Q functions $Q^A(s_t, a_t)$ and $Q^B(s_t, a_t)$ working together to update the Q values, where $Q^A(s_t, a_t)$ is able to learn from the experiences of $Q^B(s_t, a_t)$ and vice versa. The update equations of Double Q-learning are [19]:

$$\begin{aligned} Q^A(s_t, a_t) = {} & Q^A(s_t, a_t) \\ & + \alpha(r_t + \gamma Q^B(s_{t+1}, \arg\max_a Q^A(s_{t+1}, a)) \\ & - Q^A(s_t, a_t)) \end{aligned} \tag{2}$$

$$\begin{aligned} Q^B(s_t, a_t) = {} & Q^B(s_t, a_t) \\ & + \alpha(r_t + \gamma Q^A(s_{t+1}, \arg\max_a Q^B(s_{t+1}, a)) \\ & - Q^B(s_t, a_t)) \end{aligned} \tag{3}$$

where next state selection for Q^A uses the Q values from Q^B and vice versa. This approach has been shown [19] to cause the algorithm to underestimate rather than overestimate the solution with a positive bias. In conditions where Q-learning performs poorly, double Q-learning has been shown to converge to the optimum solution [19].

In addition to maintaining two Q tables and updating them using the Bellman equations (2) and (3), Double Q-learning must implement a mechanism to choose an action based on the two Q tables. Our AoA detection algorithm using Double Q-learning can be seen in Algorithm 2. Like algorithm 1, the same hyper-parameters are initialized first. Two Q tables Q^A and Q^B are initialized to zeros. The same ϵ-greedy approach and reward r_t calculations are also used in Algorithm 2. To choose an action, in lines 18-19, we sum the two tables into an aggregate table Q^{AB} and pick the action that results in the maximum Q value in the Q^{AB}. To integrate the insights from the two Q tables, in lines 22-26, we randomly chose to update either one of the two Q tables with equal probability. Finally, we use the same convergence criterion of Algorithm 1.

Q-learning based algorithms have to train for a certain number of iterations. Based on the number of iterations, the algorithm may or may not converge to the solution. This makes selecting the number of iterations non-trivial. To address this issue and devise a stopping criterion, we calculate CoV after a certain number of RSS samples, *tickCount*. CoV is a statistical measure of how dispersed data samples are from the mean of the sample space. It is the ratio of standard deviation and mean of a certain number of samples. In our algorithm, the required number of samples to calculate CoV is initialized to $tickCount \in (5, 30)$. The array *RSS* is used to store RSS samples as the algorithm is training on the fly. If the counter *sampleCount* is greater than or equal to the *tickCount*, then enough samples have been collected to calculate the CoV. While the CoV is greater than or equal to the selected *Threshold* value, the algorithm will continue to train. When CoV is less than or equal to the *Threshold*, the convergence criterion is met and the algorithm breaks out of the while loop. The smaller CoV means that the RSS samples have stabilized and it is safer to stop the algorithm and return the last steering angle as the detected AoA.

B. DEEP Q-LEARNING

Traditional Q-learning relies on a Q-table to assign Q values to state-action pairs (s_t, a_t). Q-learning is advantageous for solving less complex problems with a small state-action space. However, in complex settings involving a large number of (s_t, a_t) pairs, using Q-tables may not be energy efficient or even may be infeasible. To address this challenge, Deep Q-learning was introduced, a.k.a. Deep-Q-Network (DQN) [55]. Unlike conventional Q-learning, DQN does not rely on a Q-table, but rather employs a neural network to estimate the Q values for given actions. Figure 13 depicts of neural networks used in a DQN. State values are fed into the neural networks. The Q-network is used to determine the value of $Q(s_t, a)$ from the Bellman equation, while the target Q value $Q(s_{t+1}, a)$ is predicted using the target network. The orange nodes in Figure 13 represent the neurons of the neural network. At each node, state values are multiplied by given

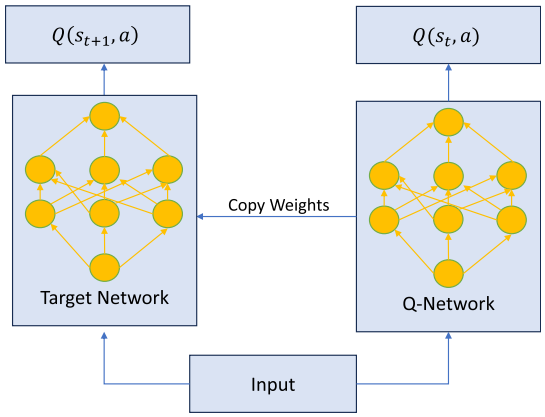


FIGURE 13. Deep Q-learning neural networks.

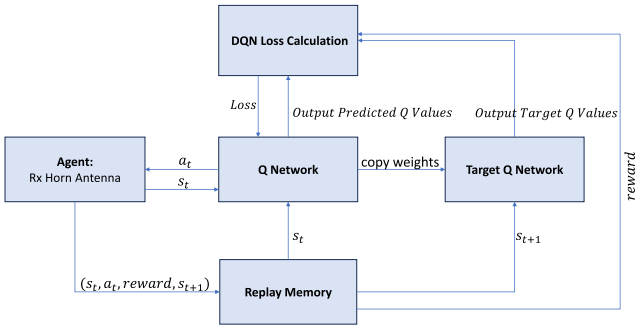


FIGURE 14. DQN architecture.

weights and summed. The resulting sum is then passed to an activation function that is used to determine whether to switch the neuron on or off. Numerous activation functions are used in neural networks, such as Sigmoid, Hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU).

The ReLU function is a piece-wise function defined by $f(x) = \max(0, x)$. Since the function is using the max value of x, all negative values are by default zero. ReLU is one of the most popular activation functions, used mainly in CNNs. Unlike Sigmoid and tanh, ReLU does not suffer from the vanishing gradient problem, which leads to saturation. Recently in [16], a neural network was used to detect the beam direction of a four element circular antenna array. The performance of activation functions ReLU, Leaky ReLU, and tanh were compared, and ReLU outperformed the others for detecting the azimuth and elevation angle of the antenna beam. For our DQN setup, we employ ReLU activation functions in our hidden layers.

The DQN architecture can be seen in the block diagram presented in Figure 14. The agent takes actions using the ϵ-greedy policy. After taking an action, the sample, which is the tuple of <s_t, a_t, r_t, s_{t+1}>, i.e., <current state, action, reward, and next state>, is stored in memory for later use in training. It is recommended to use a 32 or more samples for training, for which we used a batch size of $N = 64$.

Random batches of data samples are selected for network training. The current state s_t is fed into the Q network. The Q network outputs predicted Q values with respect to all possible actions. The predicted Q value $Q_{predicted}$ corresponding to the sampled action a_t is selected. The next state s_{t+1} value from the sample is fed into the target network. The maximum Q value from the output of the target Q network and reward from the selected sample are used to compute the target Q value Q_{target}, as follows:

$$Q_{target} = Reward + \gamma \max_a(Q_1, Q_2, ..Q_n, a) \quad (4)$$

$$Loss = MSE(Q_{predicted}, Q_{target}) \quad (5)$$

where the loss is computed by taking the mean squared error (MSE) between the predicted and target Q value in (5). The loss is back-propagated to improve the weight values of the $Q_{network}$. With more accurate weights the Q network can predict more accurate Q values. After training the Q network weights are aligned to the target Q network. The process is then repeated for every episode.

Algorithm 3 AoA Detection With DQN

Input: $\epsilon \in (0, 1)$, the greediness policy; *tickCount* $\in (5, 30)$, the number of RSS samples maintained for implementing convergence criterion; *Threshold* $\in (0.01, 1.5)$ threshold for convergence criterion;
Output: *detectedAoA*, AoA detected by the algorithm.
Lines 3-10 from Algorithm 1
$N \leftarrow 64$ ▷ Batch size (samples)
$D \leftarrow N$ ▷ Replay memory initialized with batch size N
$T \leftarrow 10$ ▷ Update rate (samples/update) for aligning weight values
Initialize $Q_{network}$ and Q_{target} with random weights
while *Threshold* $\leq$ *CoV* **do**
  $RSS.append(RSS_t)$ ▷ Store RSS data in array
  if Uniform(0,1)< ϵ **then**
    $a_t \leftarrow$ round(Uniform[0,1])
  else
    $Q_{predicted} \leftarrow Q_{network}.predict(s_t)$ ▷ Predicted Q values from Q network using current state
    $a_t \leftarrow \max_a Q_{predicted}(s_t, a)$
  end if
  Lines 19-35 from Algorithm 1
  Append the new sample (s_t, a_t, r_t, s_{t+1}) to the replay memory D
  if $length(D) > N$ **then** ▷ Begin training when batch size is met
    retrain() ▷ Call function retrain() to train DQN
  end if
  if *sampleCount*%T==0 **then** ▷ When update rate is reached copy weights to target network
    $Q_{target}.weights \leftarrow Q_{network}.weights$
  end if
  Lines 37-40 from Algorithm 1
end while
return s_t

Our DQN implementation's pseudo-code is presented in Algorithm 3. At the beginning, the hyper-parameters γ and ϵ are initialized. The batch size D is initialized to a specific size N. The update rate is set to T=10, which is less than the batch size and should be sufficiently fast as prior studies showed $T > 100$ being very successful for various settings [56], [57], [58]. Both the Q and target Q networks are initialized to random weight values. Like our previous algorithms, the current state s_t is initialized to a random angular value, based

off our experimental scenario. For the 90° experimental setup the random angle can be any value in [80,100] degrees and [120,140] degrees for the 130°. An outer while loop comparing the *Threshold* value and *CoV* is used to decide the stopping condition for the algorithm until the threshold value is met.

The agent performs an action based on the ϵ-greedy policy while taking an action. If ϵ is less than a random number, the agent takes an action that results in the maximum predicted Q value $Q_{predicted}$. If the batch D is greater than N the network training begins. The retrain() function is called to initiate network training, during which the operations in (4) and (5) are applied. As a result, the Q network weights are improved via training. The Q network weights are then copied onto the target Q network at a rate of T.

C. COMPLEXITY OF RL-BASED AoA DETECTION

Our RL-based AoA detection methods have differences in terms of computational and memory complexity. Depending on the resources available at the receiver where these algorithms will be running, these complexities may be important. These algorithms work with a window of RSS samples (represented by *tickCount* in Algorithms 1 and 2) when testing the convergence criterion. Let $W = tickCount$ indicate the number of RSS samples being maintained, S indicate the number of states, and A indicate the number of actions at each state. The Q-learning and Double Q-learning algorithms will calculate the new convergence criterion over the W samples and find the maximum Q value across possible actions two times, which requires $W+2A$ steps to calculate the next action to take. Hence, their time complexity is $O(W + A)$. Their memory needs will be dominated by the Q table size and the RSS samples being maintained, which amount to $SA+W$ and $2SA+W$ for Q-learning and Double Q-learning, respectively. Hence, both algorithms have a worst-case memory complexity of $O(SA + W)$.

DQN does not use Q tables but a batch of samples (D with size N in Algorithm 3) and two neutral networks (Q_{target} and $Q_{network}$ in Algorithm 3) to perform learning. The size of a neural network in DQN is dominated by the number of layers and the number of neurons per layer. For a neural network with L layers and K neurons per layer, there are $2K + (L-1)K^2$ weights since there will be K input and K output weights, and K^2 weights in between two consecutive layers. After the batch of samples is full, DQN updates the neutral network weights during retrain (line 19 in Algorithm 3) when determining the next action to take. At every T samples, it updates the target neural network Q_{target} (line 22). In the worst case, these neutral network operations can take $2K + (L-1)K^2 + (2K + (L-1)K^2)/T$. Factoring in the action selection (line 14), the worst-case time complexity of determining next action in DQN is $O(A+2K+(L-1)K^2+(2K+(L-1)K^2)/T) = O(A + 2K + (L-1)K^2) = O(A + LK^2)$. Similarly, the memory needed for DQN is dominated by the batch size N and the neural network size. Considering the window of RSS samples needed for convergence criterion implementation, the worst-case memory complexity is $O(N+2K+(L-1)K^2+W) = O(N + W + LK^2)$.

TABLE 1. Computational and memory complexity of RL-Based AoA detection algorithms.

RL-Based AoA Detection	Computation Complexity (s)	Memory Complexity (bits)
Q-learning	$O(W+A)$	$O(SA+W)$
Double Q-learning	$O(W+A)$	$O(SA+W)$
DQN	$O(A+LK^2)^*$	$O(N+W+LK^2)$

*Assuming $L > 1$. $O(A+K)$ when $L = 1$.

Table 1 shows the worst-case time and memory requirements of the three AoA detection algorithms in our study. The key benefit of DQN is it memory requirement is independent of the state-action space size, SA, which makes it a perfect match for designing more complex state-action space designs. In our designs, we kept the state-action space to a minimum, i.e., $S = 180$ and $A = 2$, since we have used Horn antennas which can only be mechanically steered left or right. However, for phased-array mmWave antenna systems, state-action space can be quite large as they allow switching the beam electronically from any beamforming configuration to any other possible. The benefit of Q-learning and Double Q-learning is their small time complexity, which is independent of the state space. DQN, however, requires time complexity quadratic with K when $L > 1$. Hence, size of the neural network, which is essentially used to learn the patterns in the state-action space, increases the time complexity of DQN. In our design, we used $K = 20$ and $L = \{1, 5\}$, which did not cause any issues and the DQN implementation was able to calculate the next action well before the `sleepTimer` was off. However, for fast-responding antenna systems like phased-arrays, the DQN implementation will need to be carefully tuned to make sure it does not become a bottleneck in the online process of measuring the RSS and deciding the next action.

D. EXPERIMENTAL EVALUATION AND RESULTS

To understand the efficacy of our RL-based AoA detection Algorithms 1, 2, and 3, we measured the average AoA error and average convergence time for different hyper-parameters as the stopping criterion *Threshold* varies. The *Threshold* value determines how strict the convergence criterion is. Hence, smaller *Threshold* causes the AoA detection algorithms to search the AoA for a longer period of time. This enables them to find a more accurate AoA. Hence, there is a trade-off between the error in AoA estimate and the convergence time of the algorithms. In terms of hyper-parameters, we tried different (ϵ,α) combinations for Q-learning and Double Q-learning, and different ϵ values for DQN.

1) EXPERIMENTAL SETUP

We considered two scenarios for comparing the performance of the Algorithms 1, 2, and 3. As seen in Figure 15, the range between the transmit and receive horn antennas is 1.5 ft.

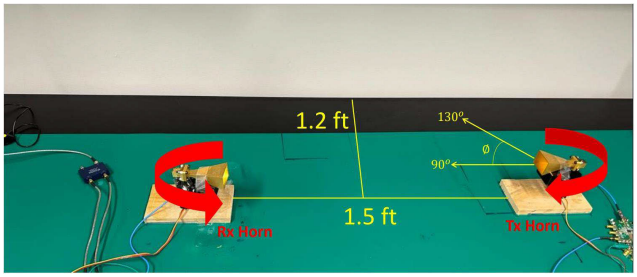


FIGURE 15. Experimental setup.

The distance to the wall to the center is 1.2 ft. In the first experiment scenario, the transmit antenna is fixed to 90°, pointing towards the receive antenna to compose an LoS path to the receiver. The receive horn antenna is initialized to a random angular state value between 80 and 100 degrees. For the second scenario, the transmit horn antenna is rotated 130° to the right, composing a NLoS path to the receiver. The main lobe of the transmitted signal is reflected off the wall. The initial angle is set to a random value between 120 and 140 degrees.

For Q and Double Q hyper-parameters ϵ and α are set to 0.1, 0.2, 0.3, and 0.4. Likewise, ϵ was set to the same values in our DQN algorithm. Unlike Q and Double Q, DQN does not use the learning rate α, since it uses neural networks to estimate Q values from the Bellman's equation. For our Q and Double Q algorithms, hyper-parameters were varied to measure which combinations of (ϵ, α) resulted in the best performance with respect to both AoA detection and convergence time. Using the same procedure, we compare the performance of DQN to Q and Double Q learning. We train our DQN model using 20 neurons for the hidden layers of the neural network. Further, we compare the performance of using a single layer neural network against 5 hidden layers. The average AoA error and convergence time was measured for 13 threshold values from 0.01 to 1.5 in increments of 0.1. This was done using Q, Double Q, and DQN training for both the 90° and 130° experimental scenarios.

As a baseline we implement a simple AoA detection algorithm (detailed in Algorithm 4) and compare the RL-based algorithms to it. Like the RL-based algorithms, the horn antenna is initialized to a random angle in [80,100] for the 90° degree scenario and [120,140] for the 130° scenario. Then, we randomly choose to move the antenna to left or right (lines 3-9). If there was a decrease in measured RSS for the chosen direction, we choose to go the opposite direction (lines 10-12). Then, we continue to steer the antenna in the chosen direction until we observe a decrease in the measured RSS. This algorithm is a greedy search mechanism and stops at the first local optima, minimizing the AoA detection time.

2) CONVERGENCE CRITERIA

A key issue with RL is deciding when to stop as there is no ground truth or a-priori training. After running the experiments, we analyzed the observed RSS values and the states the RL algorithms navigated during their search, or training,

Algorithm 4 Baseline Algorithm

```
1:  Randomize currentAngle                    ▷ Initialize to a random angle
2:  previousRSS                               ▷ Measure RSS at currentAngle
3:  if Uniform(0,1) < 0.5 then
4:      currentAngle + +                      ▷ Turn left
5:      TurnAngleLeft ← True
6:  else
7:      currentAngle − −                      ▷ Turn right
8:      TurnAngleLeft ← False
9:  end if
10: if currentRSS < previousRSS then
11:     TurnAngleLeft ←!TurnAngleLeft         ▷ Turn to the opposite
            direction
12: end if
13: previousRSS ← currentRSS
14: while True do
15:     if TurnAngleLeft then
16:         curentAngle + +                   ▷ Turn left
17:     else
18:         currentAngle − −                  ▷ Turn right
19:     end if
20:     if currentRSS < previousRSS then
21:         break                             ▷ Break out of while loop
22:     end if
23:     previousRSS ← currentRSS
24: end while
```

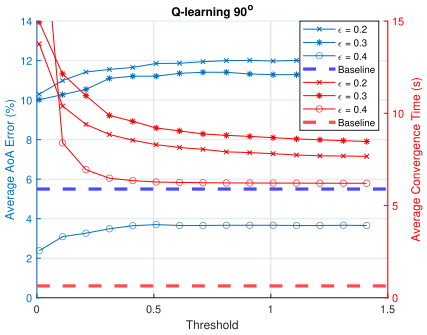


FIGURE 16. 90° LoS: Q-learning.

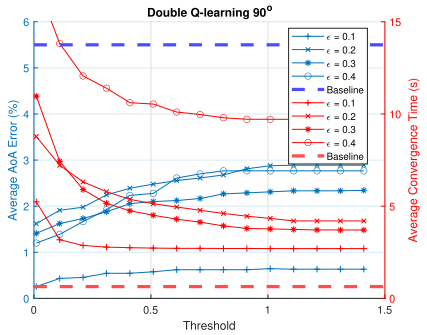


FIGURE 17. 90° LoS: Double Q-learning.

for the best AoA. We tried to find the best convergence criteria. To this end, we applied the stopping condition on the CoV of the RSS values, as detailed in Algorithm 1, such that CoV is less than a *Threshold* for *tickCount* times contiguously.

To calculate the CoV, we applied Exponentially Weighted Moving Average (EWMA) on the average and standard deviation of RSS with weights γ and β, respectively. The EWMA weights γ and β can be any value in (0,1) and are multiplied

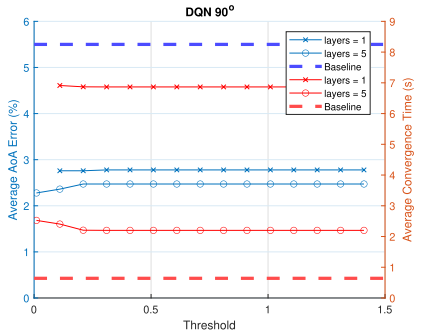


FIGURE 18. 90° LoS: DQN.

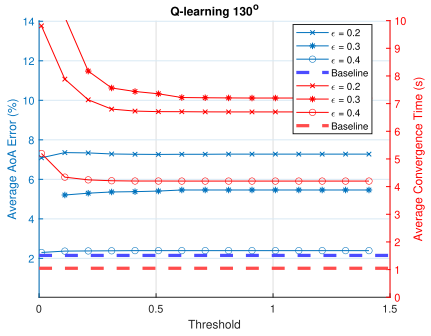


FIGURE 19. 130° NLoS: Q-learning.

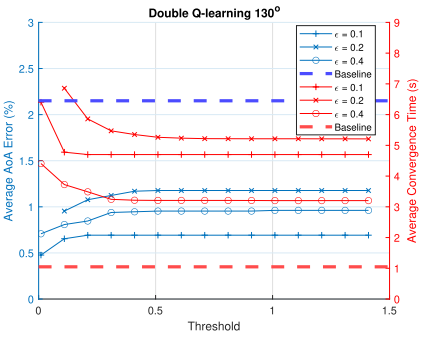


FIGURE 20. 130° NLoS: Double Q-learning.

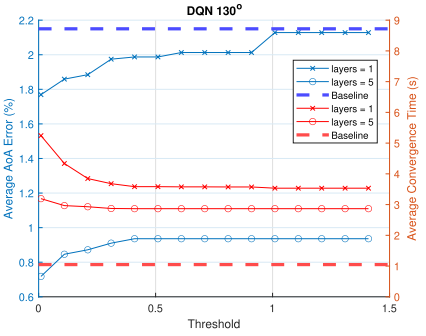


FIGURE 21. 130° NLoS: DQN.

with the latest RSS sample with $1-\gamma$ and $1-\beta$ being multiplied with the latest moving average value. Higher γ and β means the latest RSS sample gets more weight in the moving average of the average and standard deviation. Hence, to obtain a good balance, the EWMA weights must be tuned. To calculate the predicted AoA, we applied EWMA on the state s_t at which the receiver antenna is positioned over time. For this purpose, we used the same weight, γ, as the RSS moving average, since RSS should be directly related to the receiver antenna's angular position, i.e., the state s_t. Using this predicted AoA, we calculated the AoA detection error, i.e., the difference between the predicted AoA and the true AoA.

When tuning the EWMA weights, γ and β, we focused on minimizing the convergence time as *Threshold* increases. Intuitively, when *Threshold* on the CoV increases the learning algorithm will stop earlier as it is easier to satisfy the stopping condition of $CoV < Threshold$. Likewise, the AoA detection error should increase and converge to a value as *Threshold* increases. To capture this, we searched for the best γ and β combination, each being in (0.1,0.9) with step size 0.1, that yields the quickest and smoothest convergence of the AoA detection error. To do so, we minimized the total reduction on the AoA detection error as *Threshold* increases. As expected, this tuning process yielded different γ and β values for each hyperparameter combination of the learning algorithms, i.e., ϵ and α for Q and Double Q learning, and ϵ for DQN. Our analysis found that the best average (γ, β) values are (0.69, 0.73), (0.60, 0.72), and (0.55, 0.70) for Q-learning, Double Q-learning, and DQN.

Another factor that impacts the convergence efficiency is the *tickCount*, which is the number of times $CoV < Threshold$ must satisfy for the learning algorithm to stop. Hence, higher *tickCount* yields a more stringent stopping condition, implying a longer convergence time and lower AoA detection error. Our analysis found that the best average *tickCount* values are 33, 22, and 22 for Q-learning, Double Q-learning, and DQN, respectively.

3) AoA DETECTION ERROR AND CONVERGENCE TIME

To assess the performance of our RL algorithms, we varied the value of *Threshold* to examine the behavior of AoA error and convergence time. Figures 16-18 show the average AoA error and convergence time for the three RL algorithms in the 90° LoS case. Similarly, Figures 19-21 show the same the 130° NLoS case. For Q-learning and Double Q-learning, we averaged AoA error and convergence time for each ϵ value with respect to the learning rate α values used for training. For DQN, we calculated the average AoA error and convergence time over the four ϵ values for each layer count of 1 and 5. The former shows us how greediness of the Q-learning impacts the outcome, while the latter shows the impact of the DQN's depth on learning efficacy. Overall, the results verify our intuition that lower *Threshold* values result in more accurate AoA detection but slower convergence time, and larger *Threshold* values lead to less accurate AoA detection but faster convergence time.

As expected, our baseline method in Algorithm 4 was able to detect AoA faster than the RL-based methods but with a notable loss in accuracy. The baseline algorithm was averaged over 25 episodes. For the 90° scenario, the baseline algorithm has an average AoA error of 5.5% and convergence time of 0.64 s. For the 90° scenario, it was able to outperform Q-learning with ϵ values of 0.2 and 0.3. However, with an

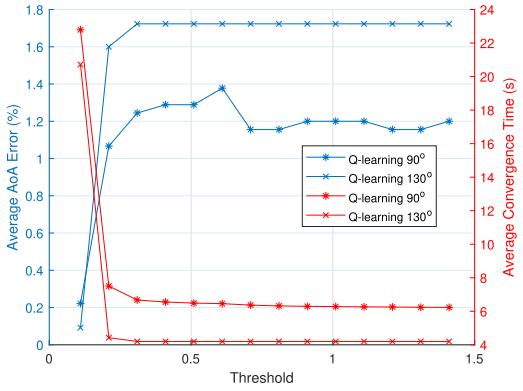
ϵ value of 0.4, Q-learning has more accurate AoA detection. Both double Q-learning and DQN outperformed our baseline algorithm with respect to AoA accuracy, but with a slower convergence time. The baseline algorithm for the 130° scenario has an average AoA error of 2.15% and a convergence time of 1.048 s. Likewise, for the 130° scenario, the baseline outperforms Q-learning with ϵ values of 0.2 and 0.3. The accuracy in AoA detection is similar for Q-learning with an ϵ value of 0.4. Similarly, double Q-learning and DQN result in more accurate AoA detection, but with a slower convergence time.

As expected, in Q-learning, higher ϵ (i.e., less greedy policy) gives result to a more accurate AoA detection as well as lower convergence time. For both the 90° LoS and 130° NLoS scenarios, the ϵ value of 0.4 resulted in the most accurate detected AoA and faster convergence time compared to smaller values of ϵ of 0.2 and 0.3. In Q-learning, we could not obtain consistently convergent results (and, hence, not shown) when ϵ is set to 0.1, clearly showing that Q-learning cannot perform well when it is forced to exploit too quickly.

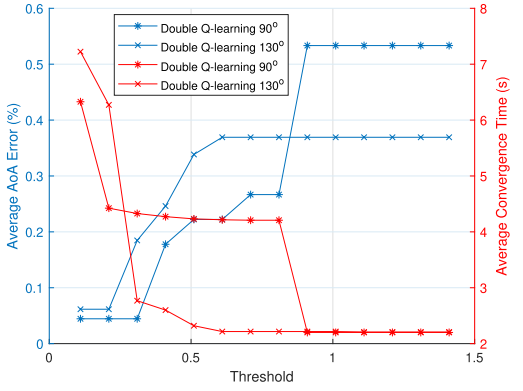
In Double Q-learning, we observed that there is a more complicated trade-off about greediness. For the 90° LoS scenario, the algorithm attained similar AoA detection error when ϵ is 0.1 and 0.4. As expected, the convergence time is significantly higher when ϵ is 0.4. This means the best outcome happens, for Double Q-learning, when ϵ is set to 0.1. When we look at the 130° NLoS scenario, however, we can clearly see that $\epsilon = 0.4$ is the winner. The reason for this outcome is that the 90° LoS scenario is simpler than the 130° NLoS scenario, which allows an exploitative setting (i.e., $\epsilon = 0.1$) for the algorithm attains better outcome. It is noteworthy that we could not obtain convergent results for the 130° NLoS scenario when ϵ is set to 0.3, which again points to the interaction taking place between the difficulty of learning the scenario and the goals of attaining low AoA detection error in short time.

Increasing the depth of the neural network in DQN yields better results. The plots presented in Figures 18 and 21 are the average AoA error and convergence time for DQN in the 90° LoS and 130° NLoS scenarios, respectively. These results clearly show that increasing the number of layers attains more accurate AoA detection as well as faster convergence time. In the 90° LoS scenario, increasing the layer count from 1 to 5 improves DQN's performance around 11% (from 2.8% to 2.5%) and 68% (from 6.9 s to 2.2 s) in terms of average AoA error and convergence time, respectively. Likewise, for the 130° NLoS scenario, increasing the layer count improves DQN's performance around 57% (from 2.1% to 0.9%) and 20% (from 3.5 s to 2.8 s) in terms of average AoA error and convergence time, respectively.

Based on the average results, DQN outperforms Q-learning with respect to both AoA detection accuracy and convergence time. In comparison to DQN, Double Q-learning attains more accurate AoA detection in similar convergence times. However, this requires proper tuning of its parameters. For example, when $\epsilon = 0.1$ for the 90° LoS scenario (Figure 17),



(a) Q-learning



(b) Double Q-learning

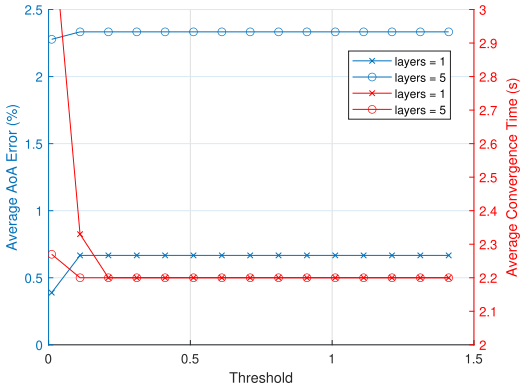
FIGURE 22. Results minimizing error and convergence time for Q-learning and double Q-learning.

Double Q-learning attains 0.62% AoA error in about 2.7 s. Comparatively, with 5 layers, DQN's best accuracy is around 2.47% in about 2.2 s (Figure 18). Similarly, when $\epsilon = 0.4$ for the 130° LoS scenario (Figure 20), Double Q-learning attains 0.95% AoA error in about 3.2 s. Comparatively, with 5 layers, DQN attains around 0.93% AoA error in about 2.8 s (Figure 21). Increasing the layer count for DQN attains faster convergence times. However, this requires more memory for DQN.

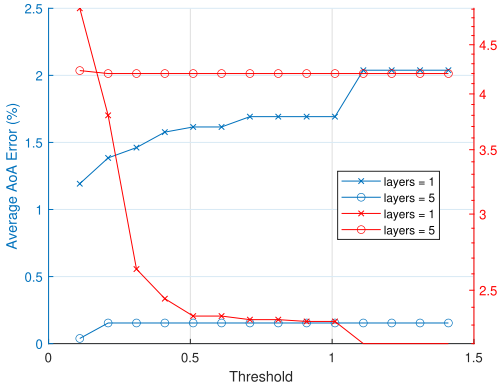
Overall, a key takeaway is that the trade-off between AoA detection error and convergence time must be balanced, which we will delve into in next.

4) BALANCING AoA DETECTION ERROR AND CONVERGENCE TIME

In order to balance the trade-off between more accuracy in AoA detection and faster convergence, we exhaustively search the hyper-parameter combinations of the RL algorithms. For a fixed *Threshold* value, we find the best hyper-parameter combination by minimizing both the AoA error and convergence time at the same time. To do so, we search for the combination that minimizes the product of the two metrics, i.e, Average AoA Error and Average Convergence Time. In particular, for Q- and Double Q-learning,



(a) DQN 90^o



(b) DQN 130^o

FIGURE 23. Results minimizing error and convergence time for DQN.

we solve the following problem:

$$\arg\min_{\epsilon,\alpha} \text{<Avg. AoA Error>} * \text{<Avg. Convergence Time>} \tag{6}$$

across all (ϵ, α) combinations that yielded a convergent result in the previous section. DQN doesn't use learning rate α. Therefore, we search for the ϵ value that minimized the product the two metrics for the various neural network layers used for training the algorithm. That is, we solve the following problem for DQN

$$\arg\min_{\epsilon} \text{<Avg. AoA Error>} * \text{<Avg. Convergence Time>}. \tag{7}$$

In the above optimizations, minimizing the product of error and convergence time enables us to find a balanced setting. However, it is possible to give more weight to one of these metrics as part of a weight geometric mean. We assume equal weight between the error and convergence time.

Joint minimization of the AoA error and the convergence time results in improvement on both target metrics. Figure 22 presents the outcome of solving (6) for Q-learning. The joint minimization of the AoA detection error and the convergence time still follows the trade-off, i.e., lower *Threshold* values result in more accurate detected AoA but take longer to converge. Compared to the average values presented in Figures 16 and 19, the joint minimization of the two metrics results in more accurate AoA detection, which quite encouraging as it means the joint minimization effectively does not sacrifice from the accuracy. We observe similar outcomes for Double Q-learning. When compared to the average values in Figures 17 and 20, the joint optimization gives better results, and the AoA error is consistent with less than 1% across all *Threshold* values. When comparing the joint minimization results in Figures 22(a) and 22(b), Double Q-learning is superior to Q-learning in terms of both metrics. Likewise, as seen in Figure 23, DQN also outperforms Q-learning for both accuracy of the detected AoA and the convergence time. Both Double Q-learning and DQN produced AoA error less than 1% and convergence times of less than 5 seconds. Overall, the five-layer DQN algorithm outperformed the other algorithms with respect to AoA error and convergence time.

A key challenge of RL algorithms is to find a good hyper-parameter combination that can work well across many cases. As observed thus far, different parameter combinations perform the best for each algorithm in the two scenarios. To obtain a high-level understanding of which parameter values tend to perform the best, we use heat maps to visually depict to find the best performing values of the parameters. Thus, in Figure 24, we plotted heat maps of the (α, ϵ) combinations for Q-learning and Double Q-learning, and the $(\epsilon$, layers) combinations for DQN. For Q-learning, the ϵ value of 0.4 is the obvious choice while the best α turns out to be 0.2 and 0.4 for the 90° LoS and 130° NLoS scenarios, respectively. For Double Q-learning, an ϵ value of 0.1 with α of 0.3 and 0.4 are the best performing combinations for the 90° LoS scenario. However, for the 130° NLoS scenario, the best choice of ϵ is 0.4 and α is 0.2. This indicates that the potential gain from Double Q-learning is more dependent on careful setting of its parameters. Finally, for the single layer DQN, an ϵ value of 0.1 is prevalent while ϵ values of 0.4 and 0.3 are prevalent for five layers. This seems to indicate that attaining the best values from DQN configurations with deeper neural networks requires increasing the algorithm's exploratory behavior (i.e., higher ϵ), which means it will likely take longer to converge.

5) LONGER RANGE AND BLOCKAGE

To evaluate the robustness of the algorithms, we increased the range between the transmit and receive antennas to approximately 8 ft for the 90° scenario. As seen in Figure 25, we placed a brick on the perfect alignment angle between the transmit and receive antennas to establish a more complex channel. Further, the bench frame is also causing blockage to the field of view (FOV) of the receive horn antenna. These blockages constitute a channel in which the receive angle corresponding to the peak RSS is harder to determine. We evaluate the performance of the RL algorithms with a γ value of 0.01. With a small value of γ the agent will be more inclined to consider immediate rewards, rather than future ones.

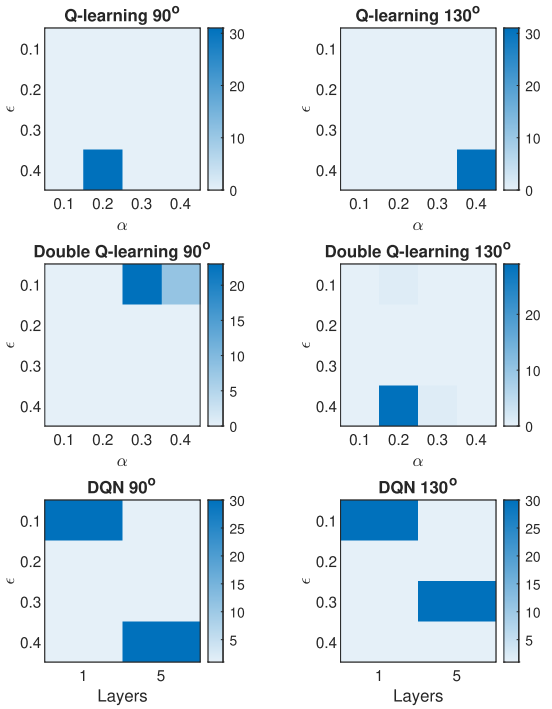


FIGURE 24. Heat map of the best performing parameter combinations.

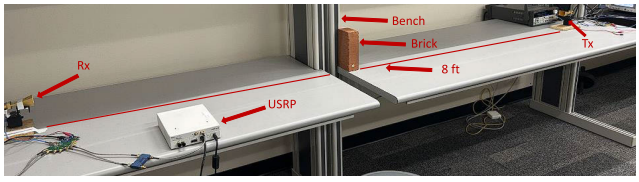


FIGURE 25. Blockage set-up.

We compare the performance of the algorithms using the fine-tuned hyper-parameter values from Figure 24. We use (ϵ, α) combination of (0.4,0.2) for Q-learning and (0.1,0.3) for Double Q-learning. For DQN, we use an ϵ value of 0.1 for a neural network with five hidden layers. To improve the performance of our DQN, we increase the number of neurons per layer to 64. Figure 26 shows a comparison of the average AoA error and convergence times as the threshold varies. Like the previous results, both Double Q-learning and DQN outperform Q-learning. As before, the AoA detection and convergence times are comparable between Double Q-learning and DQN. Double Q-learning attains an AoA error of 0.88% with a convergence time of 2.2 s. Likewise, DQN attains an AoA error of 0.88% at a convergence time of 2.5 s. In comparison to the simple 90° scenario with 1.5 ft range and no blockage (Figures 22a, 22b, and 23a), the algorithms perform very similarly.

V. CONCLUSION AND FUTURE WORK

Using commercially available hardware, we presented an affordable mmWave SDR testbed platform. Our testbed uses open source tools such as GNU Radio and the Python high-level programming language to enable RL-based ML algorithms such as Q-learning, Double Q-learning, and

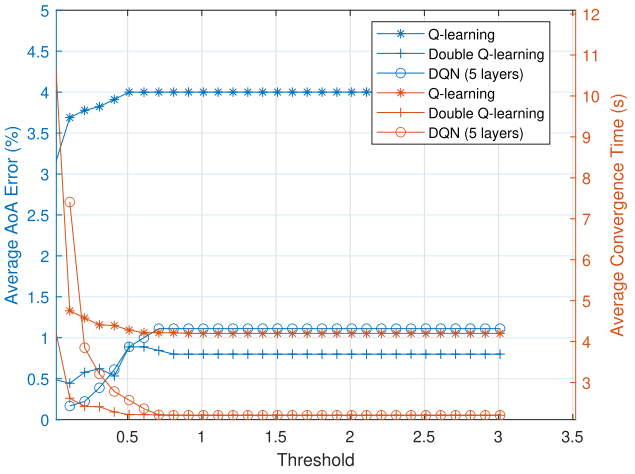


FIGURE 26. Experimental setup with blockage.

DQN. In order to manage the timescale difference between mmWave antennas' beamforming speed and the ML algorithms' execution speed, we devised a multi-threaded design where threads for antenna monitoring, ML algorithm, and GNU Radio work together to make sure measurements from the underlying antenna system are correct. To provide a convenient API to the ML algorithms, we designed and implemented directional programming functions, e.g., `setAngle()`, that encapsulate the beamforming details of the underlying antenna system.

We designed new AoA detection algorithms and tested them on our testbed. Using RL, we achieved within 1° of AoA detection in LoS and NLoS mmWave links. Our study investigated the best combinations of hyper-parameters, ϵ for greedy policy and α for learning rate, that minimize the AoA detection error and convergence time for Q-learning and Double Q-learning. For DQN, we investigated the greedy policy ϵ that minimizes the AoA detection error and convergence time for the algorithm. Our work shows that both Double Q-learning and DQN outperform Q-learning. While DQN may not be necessary due to the simplicity of our state-action space, we demonstrated that our testbed can employ deep learning for real-time experimentation. We plan on adapting DQN for more complex mmWave patch antenna array systems that can perform 3D beam steering with respect to azimuth and elevation. Unlike our current setup which can only steer the antenna either left or right, an antenna array enables the agent to take actions based on the number of available beams. For example, in [59], the patch antenna array system can form sixty-two beam patterns, giving the agent sixty-two possible choices to select from. With Q-learning, a Q-table is used to store $Q(s_t, a_t)$ values at each iteration for a given action. With sixty-two possible actions, Q-learning may not be a suitable choice for training the agent. For a larger state-action, it would be more efficient to use DQN, which employs a neural network to determine the Q values for all potential actions.

There are several future directions for our study. First, our testbed can be improved in terms of its antenna hardware

and enable richer set of mmWave beamforming capabilities. It is feasible to expand our testbed to a MIMO arrangement by incorporating extra USRP units and antennas. Our setup uses a single USRP model N210, equipped with one transmit and one receive port. The USRP N210 features a MIMO expansion port, facilitating connection to another USRP N210 via an adapter. By adding another USRP N210, a 2×2 MIMO system with two transmit and two receive ports can be established. However, expanding to a MIMO setup larger than 2×2 requires more hardware though our software setup can handle such larger designs. Even though our current proof-of-concept illustrates the ML-based programmable directionality as a capability, with the usage of inexpensive servos and horn antennas, our current platform setup is limited. We plan to equip our tested with phased array antenna systems, which can steer the antenna beam in the order of microseconds. This would improve our software run-time and improve the convergence time of the ML algorithms. Tuning of the `sleepTimer` for these new antenna systems will need to be performed.

In terms of AoA detection, reducing the convergence time further is of high interest. A maximum of 300 ms beam sweeping period is needed for future vehicle-to-infrastructure communication systems when the vehicle moves at an average speed of 10.5 mph [60]. While this maximum will be smaller for walking speeds, faster AoA detection enables more practical use of the mmWave communication links, e.g., it becomes possible to align the beams for better throughput. Yet, practical convergence times for AoA detection in mmWave systems is limited by the antenna response time, which is driven by the amount of time needed to switch the mmWave beam from one angle to another. Since we used a horn antenna system with mechanical steering, our system is limited by the response time of the servo motor, which is kept under control by the `sleepTimer` in the testbed. Hence, the convergence times we are reporting are based on the `sleepTimer` of 200ms. The `sleepTimer` had to be long in our experiments because of the mechanical steering involved in the setup. For phased array mmWave systems, the switch time can be in the order of microseconds. For example, patch antenna arrays [59] have enabled mmWave platforms to switch from one codebook combination (of phases) to another within microseconds (us). This will mean that the AoA detection can be done in the order of tens of microseconds. Future work on integrating such patch antenna arrays to our testbed platform will enable testing the RL-based AoA detection methods in the order of tens of microseconds. The RL algorithms will need to be adapted to work with the patch antenna arrays.

Our RL-based AoA detection algorithms utilized a state space defined by the latest beamsteering angle. It is possible to design state spaces that uses the previous steering angles and the recent RSS measurements for defining a state. A systematic evaluation of the trade-off between more state space complexity and the AoA detection needs to be explored.

Further, more complicated scenarios, e.g, involving multiple transmitters, can be explored to observe the performance of RL-based AoA detection in mmWave frequencies. In our design, the ML-based AoA detection takes place in the receiver side. An interesting direction of research would be to see if it is possible to align (not just detect AoA) and stabilize the transmitter and receiver beams when the transmitter and receiver are individually making ML-based decisions.

ACKNOWLEDGMENT

An earlier version of this paper was presented in part at the IEEE INFOCOM International Workshop on Computer and Networking Experimental Research using Testbeds (CNERT) 2023, New York, USA [DOI: 10.1109/INFOCOMWKSHPS57453.2023.10226092]; and in part at the IFIP Internet-of-Things Conference 2023, Dallas-Fort Worth, TX, USA [DOI: 10.1007/978-3-031-45878-1_11].

REFERENCES

[1] M. Jean, M. Yuksel, and X. Gong, "Millimeter-wave software-defined radio testbed with programmable directionality," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops*, May 2023, pp. 1–8.

[2] M. Jean and M. Yuksel, "Reinforcement learning based angle-of-arrival detection for millimeter-wave software-defined radio systems," in *Internet of Things. Advances in Information and Communication Technology*, D. Puthal, S. Mohanty, and B.-Y. Choi, Eds., Cham, Switzerland: Springer, 2024, pp. 151–167.

[3] A. V. Lopez, A. Chervyakov, G. Chance, S. Verma, and Y. Tang, "Opportunities and challenges of mmWave NR," *IEEE Wireless Commun.*, vol. 26, no. 2, pp. 4–6, Apr. 2019.

[4] I. F. Akyildiz, C. Han, Z. Hu, S. Nie, and J. M. Jornet, "Terahertz band communication: An old problem revisited and research directions for the next decade," *IEEE Trans. Commun.*, vol. 70, no. 6, pp. 4250–4285, Jun. 2022.

[5] K. M. S. Huq, S. A. Busari, J. Rodriguez, V. Frascolla, W. Bazzi, and D. C. Sicker, "Terahertz-enabled wireless system for beyond-5G ultra-fast networks: A brief survey," *IEEE Netw.*, vol. 33, no. 4, pp. 89–95, Jul. 2019.

[6] *5G*. Wikipedia. Accessed: Aug. 27, 2024. [Online]. Available: https://en.wikipedia.org/wiki/5G

[7] Z. Zhang et al., "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Veh. Technol. Mag.*, vol. 14, no. 3, pp. 28–41, Sep. 2019.

[8] Y. Niu, Y. Li, D. Jin, L. Su, and A. V. Vasilakos, "A survey of millimeter wave communications (mmWave) for 5G: Opportunities and challenges," *Wireless Netw.*, vol. 21, no. 8, pp. 2657–2676, Nov. 2015.

[9] T. Chen et al., "28 GHz channel measurements in the COSMOS testbed deployment area," in *Proc. 3rd ACM Workshop Millim.-Wave Netw. Sens. Syst.*, 2019, pp. 39–44.

[10] S. Yi, Y. Pei, S. Kalyanaraman, and B. Azimi-Sadjadi, "How is the capacity of ad hoc networks improved with directional antennas?" *Wireless Netw.*, vol. 13, no. 5, pp. 635–648, Oct. 2007.

[11] J. Zhang and S. C. Liew, "Capacity improvement of wireless ad hoc networks with directional antennas," in *Proc. IEEE Veh. Technol. Conf.*, vol. 2, May 2006, pp. 911–915.

[12] A. Spyropoulos and C. S. Raghavendra, "Capacity bounds for ad-hoc networks using directional antennas," in *Proc. IEEE Int. Conf. Commun.*, vol. 1, May 2003, pp. 348–352.

[13] Y. Yang, C. Sun, H. Zhao, H. Long, and W. Wang, "Algorithms for secrecy guarantee with null space beamforming in two-way relay networks," *IEEE Trans. Signal Process.*, vol. 62, no. 8, pp. 2111–2126, Apr. 2014.

[14] H. Yazdani, S. Seth, A. Vosoughi, and M. Yuksel, "Throughput-optimal D2D mmWave communication: Joint coalition formation, power, and beam optimization," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2022, pp. 1539–1544.

[15] Z. Dai, Y. He, V. Tran, N. Trigoni, and A. Markham, "DeepAoANet: Learning angle of arrival from software defined radios with deep neural networks," *IEEE Access*, vol. 10, pp. 3164–3176, 2022.

[16] G. R. Friedrichs, M. A. Elmansouri, and D. S. Filipovic, "Angle-of-arrival sensing using a machine learning enhanced amplitude-only system," *IEEE Sensors J.*, vol. 23, no. 4, pp. 3878–3888, Feb. 2023.

[17] M. M. U. Chowdhury, F. Erden, and I. Guvenc, "RSS-based Q-learning for indoor UAV navigation," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Nov. 2019, pp. 121–126.

[18] C. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, May 1992, doi: 10.1023/A:1022676722315.

[19] H. Van Hasselt, "Double Q-learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2010, pp. 2613–2621.

[20] J. Keppo, S. Rinaz, and G. Shah. (2002). *Network Pricing With Continuous Uncertainties and Multiple QoS Classes*. [Online]. Available: http://www-personal.engin.umich.edu/~keppo/QoSPricing.pdf

[21] S. Seth, M. Yuksel, and A. Vosoughi, "Forming coalition sets from directional radios," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Nov. 2022, pp. 507–514.

[22] T. Bogale, X. Wang, and L. Le, "mmWave communication enabling techniques for 5G wireless systems: A link level perspective," in *mmWave Massive MIMO*, S. Mumtaz, J. Rodriguez, and L. Dai, Eds., Cambridge, MA, USA: Academic, 2017, pp. 195–225. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128044186000091

[23] O. Abari, H. Hassanieh, M. Rodriguez, and D. Katabi, "Millimeter wave communications: From point-to-point links to agile network connections," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 169–175. [Online]. Available: https://api.semanticscholar.org/CorpusID:135479

[24] T. E. Bogale, X. Wang, and L. B. Le, "Adaptive channel prediction, beamforming and scheduling design for 5G V2I network: Analytical and machine learning approaches," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5055–5067, May 2020.

[25] B. M. ElHalawany, S. Hashima, K. Hatano, K. Wu, and E. M. Mohamed, "Leveraging machine learning for millimeter wave beamforming in beyond 5G networks," *IEEE Syst. J.*, vol. 16, no. 2, pp. 1739–1750, Jun. 2022.

[26] T. S. Cousik, V. K. Shah, J. H. Reed, T. Erpek, and Y. E. Sagduyu, "Fast initial access with deep learning for beam prediction in 5G mmWave networks," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Nov. 2021, pp. 664–669.

[27] Z. Xiao et al., "A survey on millimeter-wave beamforming enabled UAV communications and networking," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 557–610, 1st Quart., 2022.

[28] X. Gu et al., "A multilayer organic package with 64 dual-polarized antennas for 28GHz 5G communication," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2017, pp. 1899–1901.

[29] A. Şahin, M. L. Sichitiu, and I. Guvenç, "A millimeter-wave software-defined radio for wireless experimentation," 2023, *arXiv:2302.08444*.

[30] J. Zhang, X. Zhang, P. Kulkarni, and P. Ramanathan, "OpenMili: A 60 GHz software radio platform with a reconfigurable phased-array antenna," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 162–175.

[31] S. K. Saha et al., "x60: A programmable testbed for wideband 60 GHz WLANs with phased arrays," *Comput. Commun.*, vol. 133, pp. 77–88, Jan. 2019.

[32] H. Wu et al., "The tick programmable low-latency SDR system," in *Proc. 23rd Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2017, pp. 101–113.

[33] R. Zhao, T. Woodford, T. Wei, K. Qian, and X. Zhang, "M-Cube: A millimeter-wave massive MIMO software radio," in *Proc. 26th Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–14.

[34] M. Polese et al., "MillimeTera: Toward a large-scale open-source mmWave and terahertz experimental testbed," in *Proc. 3rd ACM Workshop Millim.-Wave Netw. Sens. Syst.*, Oct. 2019, pp. 27–32.

[35] J. O. Lacruz, D. Garcia, P. J. Mateo, J. Palacios, and J. Widmer, "mmFLEX: An open platform for millimeter-wave mobile full-bandwidth experimentation," in *Proc. 18th Int. Conf. Mobile Syst. Appl. Services*, 2020, pp. 1–13.

[36] I. K. Jain, R. Subbaraman, T. H. Sadarahalli, X. Shao, H.-W. Lin, and D. Bharadia, "MMobile: Building a mmWave testbed to evaluate and address mobility effects," in *Proc. 4th ACM Workshop Millim.-Wave Netw. Sens. Syst.*, Sep. 2020, pp. 1–6.

[37] A. Quadri, H. Zeng, and Y. T. Hou, "A real-time mmWave communication testbed with phase noise cancellation," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 455–460.

[38] H. Hassanieh, O. Abari, M. Rodriguez, M. Abdelghany, D. Katabi, and P. Indyk, "Fast millimeter wave beam alignment," in *Proc. Conf. ACM Special Interest Group Data Commun.* New York, NY, USA: Association for Computing Machinery, Aug. 2018, pp. 432–445.

[39] V. Va, H. Vikalo, and R. W. Heath Jr., "Beam tracking for mobile millimeter wave communication systems," in *Proc. IEEE Global Conf. Signal Inf. Process. (GlobalSIP)*, Dec. 2016, pp. 743–747.

[40] C. Zhang, D. Guo, and P. Fan, "Tracking angles of departure and arrival in a mobile millimeter wave channel," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sep. 2016, pp. 1–6.

[41] F. Zafari, A. Gkelias, and K. K. Leung, "A survey of indoor localization systems and technologies," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2568–2599, 3rd Quart., 2019.

[42] K. Wu, W. Ni, T. Su, R. P. Liu, and Y. J. Guo, "Recent breakthroughs on angle-of-arrival estimation for millimeter-wave high-speed railway communication," *IEEE Commun. Mag.*, vol. 57, no. 9, pp. 57–63, Sep. 2019.

[43] A. Alkhateeb, "DeepMIMO: A generic deep learning dataset for millimeter wave and massive MIMO applications," 2019, *arXiv:1902.06435*.

[44] M. Kaveh and A. Barabell, "The statistical performance of the MUSIC and the minimum-norm algorithms in resolving plane waves in noise," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 2, pp. 331–341, Apr. 1986.

[45] M. Pastorino and A. Randazzo, "A smart antenna system for direction of arrival estimation based on a support vector regression," *IEEE Trans. Antennas Propag.*, vol. 53, no. 7, pp. 2161–2168, Jul. 2005.

[46] O. Abari, H. Hassanieh, M. Rodreguiz, and D. Katabi, "Poster: A millimeter wave software defined radio platform with phased arrays," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw.*, Oct. 2016, pp. 419–420.

[47] *ADMV1013*. Analog Devices. Accessed: Aug. 27, 2024. [Online]. Available: https://www.analog.com/en/products/admv1013.html

[48] *ADMV1014*. Analog Devices. Accessed: Aug. 27, 2024. [Online]. Available: https://www.analog.com/en/products/admv1014.html

[49] E. M. Fennelly, *The Utilization of Software Defined Radios for Adaptive, Phased Array Antenna Systems*. Burlington, VT, USA: Univ. Vermont State Agricultural College, 2020.

[50] B. Sadhu, A. Paidimarri, M. Ferriss, M. Yeck, X. Gu, and A. Valdes-Garcia, "A software-defined phased array radio with mmWave to software vertical stack integration for 5G experimentation," in *IEEE MTT-S Int. Microw. Symp. Dig.*, Jun. 2018, pp. 1323–1326.

[51] E. Fennelly and J. Frolik, "Phase measurement and correction for software defined radio systems," in *Proc. IEEE 21st Annu. Wireless Microw. Technol. Conf. (WAMICON)*, Apr. 2021, pp. 1–5.

[52] M. Kearns and S. Singh, "Finite-sample convergence rates for Q-learning and indirect algorithms," in *Proc. Conf. Adv. Neural Inf. Process. Syst. II*. Cambridge, MA, USA: MIT Press, 1999, pp. 996–1002.

[53] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *J. Mach. Learn. Res.*, vol. 6, pp. 503–556, Dec. 2005.

[54] C. Szepesvári, "The asymptotic convergence-rate of q-learning," in *Proc. 10th Int. Conf. Neural Inf. Process. Syst.* Cambridge, MA, USA: MIT Press, 1997, pp. 1064–1070.

[55] V. Mnih et al., "Playing Atari with deep reinforcement learning," Dec. 2013, *arXiv:1312.5602*.

[56] G. Yang, Y. Li, D. Fei, T. Huang, Q. Li, and X. Chen, "DHQN: A stable approach to remove target network from deep Q-learning network," in *Proc. IEEE 33rd Int. Conf. Tools Artif. Intell. (ICTAI)*, Nov. 2021, pp. 1474–1479.

[57] S. Kim, K. Asadi, M. Littman, and G. Konidaris, "DeepMellow: Removing the need for a target network in deep Q-learning," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 2733–2739.

[58] J. F. Hernandez-Garcia and R. S. Sutton, "Understanding multi-step deep reinforcement learning: A systematic study of the DQN target," 2019, *arXiv:1901.07510*.

[59] Sivers Semiconductors. (2023). *Evaluation Kits & Evaluation Boards*. [Online]. Available: https://www.sivers-semiconductors.com/sivers-wireless/wireless-products/evaluation-kits

[60] O. Kanhere, A. Chopra, A. Thornburg, T. S. Rappaport, and S. S. Ghassemzadeh, "Performance impact analysis of beam switching in millimeter wave vehicular communications," in *Proc. IEEE 93rd Veh. Technol. Conf. (VTC-Spring)*, Apr. 2021, pp. 1–7.

MARC JEAN received the B.S. and M.S. degrees in electrical engineering from Virginia Tech, Blacksburg, VA, USA, in 2014 and 2016, respectively, and the Ph.D. degree in electrical engineering from the University of Central Florida, Orlando, FL, USA, in 2024. He has been joined the Networking and Wireless Systems Laboratory (NWSL), University of Central Florida since 2021. He is currently a Post-Doctoral Researcher at NWSL. His area of research is on millimeter wave (mmWave) antenna beam alignment using software-defined radios (SDR) platforms. While at NWSL, he designed and configured a novel SDR testbed that can transmit and receive mmWave signals over the air. His research work has won two best paper awards.

MURAT YUKSEL (Senior Member, IEEE) received the B.S. degree in computer engineering from Ege University, Izmir, Turkey, in 1996, and the M.S. and Ph.D. degrees in computer science from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1999 and 2002, respectively. He is a Professor at the ECE Department, University of Central Florida (UCF), Orlando, FL, USA, and a Visiting Scientist at MIT Lincoln Labs. He served as the Interim Chair of ECE at UCF from 2021 to 2022. Prior to UCF, he was a faculty member at the CSE Department, University of Nevada, Reno, NV, USA. He has published more than 200 papers at peer-reviewed journals and conferences. His research interests are in the areas of networked, wireless, and computer systems with a recent focus on wireless systems, optical wireless, spectrum sharing, network economics, network architectures, and network management. He is a senior member of ACM. is a co-recipient of five Best Paper, one Best Paper Runner-up, and one Best Demo Awards. He has been on the editorial boards of Computer Networks, IEEE Transactions on Communications, IEEE Transactions on Machine Learning in Communications and Networking, and IEEE Networking Letters.

XUN GONG (Senior Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from FuDan University, Shanghai, China, in 1997 and 2000, respectively, and the Ph.D. degree in electrical engineering from The University of Michigan, Ann Arbor, in 2005. He is currently a Professor of Electrical and Computer Engineering (ECE), University of Central Florida (UCF), and the Director of the Antenna, RF and Microwave Integrated Systems (ARMI) Laboratory. He joined UCF as an Assistant Professor in 2005. He worked at Air Force Research Laboratory (AFRL), Hanscom, MA, USA, in 2009, under the support of Air Force Office of Scientific Research (AFOSR) Summer Faculty Fellowship Program (SFFP). He was with the Birck Nanotechnology Center, Purdue University, West Lafayette, IN, USA, as a Post-Doctoral Research Associate in 2005. His research interests include microwave passive components and filters, sensors, antennas and arrays, flexible electronics, and packaging. He has been the recipient of the NSF Faculty Early CAREER Award in 2009. He received the UCF Reach for the Stars Award in 2016. He is awarded the UCF Lockheed Martin Professorship in 2018 to 2028. He has served on the Editorial Boards of IEEE Transactions on Microwave Theory and Techniques (TMTT), IEEE Transactions on Antennas and Propagation (TAP), IEEE Microwave and Wireless Component Letters (MWCL), and IEEE Antennas and Wireless Propagation Letters (AWPL). He was the Associate Editor of IEEE TMTT in 2018 to 2019. He served as the Associate Editor of IEEE MWCL from 2013 to 2018.