Millimeter-Wave Software-Defined Radio Testbed with Programmable Directionality

Marc Jean, Murat Yuksel, and Xun Gong
Electrical and Computer Engineering Department, University of Central Florida, Orlando, FL
marc1988@knights.ucf.edu, murat.yuksel@ucf.edu, xun.gong@ucf.edu

***Abstract*—Wireless node density and gigabit-per-second demands are pushing for more spatial reuse and higher frequency bands, which are realized by directional beamforming methods. Programming directionality of wireless beams is becoming a major need for software-defined radio (SDR) platforms. We present a low-cost "directional SDR" testbed that enables convenient programming of millimeter wave (mmWave) beam directions from a high-level programming language along with access to legacy SDR methods.**

I. Introduction

The deployment density of Internet-of-Things (IoT) devices is increasing and the gigabit-per-second wireless capacity demands are becoming expected in the emerging 5G and the envisioned 6G [1] standards. Attaining gigabit-per-second simultaneous per-user speeds in high density deployments necessitates higher spatial reuse and utilization of higher frequency bands of the electromagnetic spectrum. Both needs point to more directionality in wireless signal propagation. mmWave [2], [3] approaches are being tried for higher capacity in return of less mobility, as these higher frequency bands come with fragility to mobile and non-line-of-sight (NLOS) operation due to high path loss. Proper integration of highly *directional bands,*[1] at 22 GHz and above, into general-purpose mobile networking is an open research problem requiring joint hardware and software solutions.

Directionality offers wide bandwidth, energy-efficient, and inherently secure wireless communication. While offering higher spatial reuse and less interference, directional transceivers also 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 [4]. 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 attenuation and short range communication.

Attaining mobility at high(-rate) frequencies is challenging due to their directionality. SDR in combination with beam-steering antennas may opportunistically solve mobility and LOS challenges of directionality. If such beam-steering antennas could be controlled via SDR, very interesting family of 'software-defined beamforming' designs become possible. The ability to control the direction of signals (e.g., `setDirection()` as an API in legacy SDR platforms) with software allows PHY-MAC hybrid designs involving directional beamforming while maintaining multiple simultaneous communications with neighbors. Further, unlike hardware-based traditional tracking and acquisition techniques, such *directional SDR* enables handling of LOS detection and establishment via software.

Making *programmable directionality* a first-class citizen of SDRs requires flexible and effective integration of PHY layer beamforming capability (where wireless beams are directional) with high-level programming environments (where advanced algorithmic methods such as machine learning and data structures are available). To this end, realizing SDR-based mmWave testbeds with high programmability of the directional features of the beams while keeping the data-plane processing over field-programmable gate array (FPGA) is highly desired. In this work, we present an economical SDR testbed that enables programmable directionality in mmWave bands and measurements. The testbed utilizes a widely used Universal Software Peripheral Radio (USRP) device and provides convenience of programming directionality using Python. We show an example use of the testbed in angle-of-arrival (AoA) detection.

The rest of the paper is organized as follows. Section II describes previous mmWave testbeds from the literature and further clarifies the novelty of our work. Section III presents the architecture of our proposed system and details how USRP, GNU Radio, and Python are integrated for mmWave experimentation. Section IV presents the testbed's experimental results for an AoA detection method. Finally, we summarize our work and discuss various future research directions.

II. Related Work

Software-controlled mmWave experimentation has recently received notable attention. Most of the efforts utilized FPGA-based handling of the mmWave beams. OpenMilli [5] used FPGA-based data plane with radio frequency (RF) frontend

[1]We realize that spectrum bands themselves are not directional. However, omnidirectional beams become impractical due to significant increase in the attenuation as the signal frequency increases. Hence, we simply call spectral bands beyond 22GHz as 'directional bands'.

to access mmWave bands while allowing beam-steering on a patch antenna array. After this seminal effort, several followup studies took place in expanding mmWave experimentation capabilities using FPGA-based designs. A 12-element phased array system [6] enabled strong PHY emulation capability and studied various aspects of indoor 60 GHz links. Tick [7] added MAC experimentation capability while offering programmability using XML implemented by an FPGA. M-Cube [8] enhanced the mmWave MIMO experimentation capabilities by segregating control and data paths over multiple RF chains. The design included a USRP that guides an FPGA on the control path while a separate single data path connects to the host machine which limits the system to a single datastream. MilimeTerra [9] envisions enabling a similar capability with commercial off-the-shelf antennas.

Efforts to study the effects of mobility on mmWave systems have also received attention. Highly dynamic channel conditions arising from mobility requires software capabilities to manipulate the directions of the mmWave beams. A software implementation of beam alignment for IEEE 802.11ad used an FPGA-based baseband processing of RF signals [10]. A mobile OFDM link was demonstrated [11] using a mmWave phased array where the baseband module was switched between an FPGA and USRP to overcome USRP's limited bandwidth. The FPGA controls the phased array's weights, which limits the programmability.

The main disadvantage of the FPGA-based designs is that their programmability is limited to the hardware language. Translation of high-level programming languages to hardware languages of FPGAs 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 machine learning and multi-threaded algorithmic methods available at high-level languages, e.g., Python. The closest to our work was [12], 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 beam-steering capability controlled from a high-level programming language such as 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 [13], beam discovery and tracking [14], or AoA detection [15].

III. Testbed Architecture

The testbed design is centered around three goals, i.e., (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. 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. Fig. 1 shows a block diagram representation of the overall testbed architecture. A USRP N210 is used to transmit and receive signals via coaxial connections through the RF frontends and antennas. Fig. 2 presents a picture of the actual testbed configuration with labels mentioned in the block diagram.

A. Integration with USRP

We use UBX40 [16] as the daughterboard of the USRP, which can only process sub-6 GHz signals. Thus, the mixed signals have to be down- and up-converted for reception and transmission at the mmWave spectrum.

RF Chain. To deal with the frequency limitation of USRP devices, additional RF frontends that can process mmWave signals can be attached to the daughterboard. One approach is to connect a sequence of RF mixers, low noise amplifiers, and power amplifiers to down/up-convert the signals [17]. This approach is expensive due to the high cost of the individual microwave components. Further, connecting the sequence of devices adds more weight to the frontend and makes the testbed less flexible for mobile settings. We use off-the-shelf integrated circuit frontends, ADMV1013 [18] and ADMV1014 [19], which use Silicon Germanium (SiGe) semiconductor material resulting in a more cost-effective electronic production. These SiGe frontends provide excellent performance at mmWave bands and can up/down-convert baseband signals to/from 24-42 GHz. The frontends are programmed with Analog Devices Analysis Control and Evaluation (ACE) software. The local oscillator signals are set to 6 GHz and the ACE software is used to set the RF frontends to quadrature mode, quadrupling the local oscillator frequency.

Antennas. The testbed includes two Ka-band 15 dBi gain horn antennas, each mounted to the RF frontends. The antennas can operate with 26.5-40 GHz signals. 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 $[0^o,180^o]$. 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.

mmWave with GNU Radio. A number of software tools, including licensed ones such as MATLAB and Lab View, are available to program USRP devices. We use GNU Radio [20] to implement the signal processing blocks in the testbed.

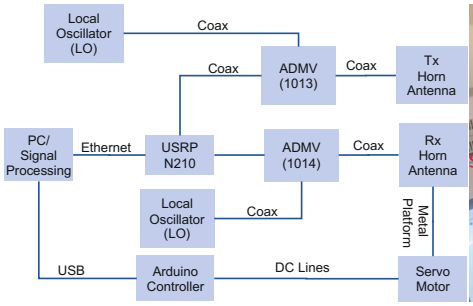


Fig. 1: Testbed architecture

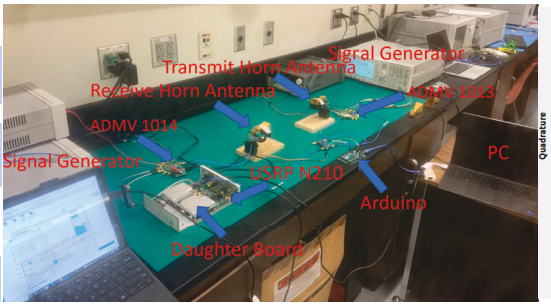


Fig. 2: Testbed setup

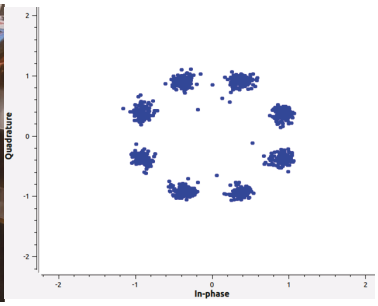


Fig. 3: 8-PSK demodulation

GNU Radio is open source and its latest versions use Python, which is heavily used in the cognitive radio community and has many advanced libraries including machine learning algorithms. GNU Radio enables tuning the Tx and Rx signals (e.g., to 2 GHz) conveniently via PC. It has built-in digital communication and signal processing blocks, such as modulation schemes, filters, and Fast Fourier Transform (FFT), 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.

GNU Radio makes it convenient to work with the USRP over its GUI as well as Python translation. Fig. 5 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 drive the signal from the transmit port of the UBX-40 daughter-board, labeled (TX/RX). The UHD:USRP Source is used to receive the signal from the receive port of the daughter-board, labeled (RX2). The two blocks communicate with the USRP N210 (via an Ethernet connection) with IP address set in the Device Address section.

The USRP N210's maximum sampling frequency is 25 MHz via 1 Gigabit Ethernet connection. With the UBX-40 daughterboard, the effective bandwidth of the testbed is 20 MHz [21]. In the setup in Fig. 5, the Signal Source block generates a 1kHz cosine signal with amplitude of 1. The sampling rate is set to 1 MHz, which is more than enough to sample a 1 kHz signal. 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 reception. Fig. 3 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 [17], [22]–[24]. 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. Our horn antenna on the receiver side can only be steered via the servo, controlled by the Arduino. GNU Radio does not have a GUI block that can interface directly with the Arduino micro-controller. Along with the antenna steering capability, we aimed to retrieve the received signal strength (RSS) so that advanced tasks such as AoA detection can be easily programmed.

```
# global variables setting up the Arduino board
board = Arduino('/dev/ttyACM0')
board.digital[10].mode = SERVO;
sleepTimer = 0.1 # seconds

def setAngle(targetAngle):
    board.digital[10].write(targetAngle);
    sleep(sleepTimer);
    return;
```

Fig. 4: Programmable directionality using a servo

1) Setting Beam Direction – `setAngle()`*:* In our testbed, the Tx horn antenna is fixed and the Rx antenna is steered by an MG995 servo, controlled directly by an Arduino board. `setAngle()` (shown in Fig. 4) implements a programmable beam direction by passing an angular value (in degrees) to the Arduino board. The first three lines import the Arduino libraries in Python. The board variable is set to the Arduino board connection at comm port ttyACM0. The third line of code is used to tie the servo to PWM Ardunio pin #10, which is used to drive the servo for angular rotation. The setAngle() function essentially sets the rotation amount in degrees by writing to the pin #10. A small sleep, `sleepTimer`, of 100ms is placed to make sure the signal reception measurements are made stably. This is needed since the servo needs some time to mechanically move the horn antenna to the `targetAngle`. Hence, a too small `sleepTimer` can cause incorrect measurements of the received signal, while a small `sleepTimer` allows faster return from the setAngle() function, providing a fast beamforming time. We will show the impact of tuning this sleep amount. The `setAngle()` as well as similar directional SDR functions can be implemented for configurable directional antennas other than the horn antennas in our testbed. The sleep amount can be tuned according to the beamswitching speed of the underlying antenna. Here, we are working with 10s of milliseconds of sleep amount but, if

the underlying mmWave antenna is a phased array system, the sleep amount can be tuned to microseconds.

2) Retrieving RSS and Multithreading: The `setAngle()` function is not hard to implement as a standalone capability. Several prior studies illustrated such beamforming capability. However, providing such a directional SDR interface along with other radio configuration and signal processing capabilities is a challenge. As one of these capabilities, we focus on retrieving RSS in GNU Radio. In order to access the RSS, we utilize a ZMQ socket in GNU Radio, shown in Fig. 5. The RSS data is pushed into the ZMQ PUSH block, which is then pulled from the ZMQ PULL block. Since the socket is located in the same PC, the IP address and TCP port assigned to the push and pull ZMQ socket is at tcp://127.0.0.1:50001.

The ZMQ data holding the RSS values can be accessed by importing the ZMQ library in Python. We implement `class RSSReader`, shown below, to pull the RSS data.

```
class RSSReader:
 keepRunning = True;
 rawReceivedData = None; #latest retrieved RSS data
 import zmq

 def __init__(self, IPAddress):
   self.maxValue = 0;
   self.context = zmq.Context()
   self.receiverSocket = self.context.socket(zmq.PULL)
   self.receiverSocket.connect(IPAddress)
   self.readerThread =
   threading.Thread(target=self.reader,)
   self.readerThread.start()

 def __del__(self):
   self.keepRunning = False #stop the reader thread
   self.readerThread.join()

 def reader(self):
    while (self.keepRunning == True):
        self.rawReceivedData = self.receiverSocket.recv()

 def readRSS(self):
    if(self.rawReceivedData != None):
        # convert to an array of floats
        float_list = array.array('f', self.rawReceivedData)
        self.maxValue = np.amax(float_list);
        return self.maxValue
    else:
        return 0;
```

The variable `rawReceivedData` is used to store the incoming RSS stream and converted to float types in an infinite while loop. The maximum value of the RSS array is returned when the `readRSS()` function is called. We chose the maximum RSS within the `rawReceivedData` because the RSS measurements in `rawReceivedData` can be noisy and picking the maximum RSS measurement is a more reliable way of measuring the actual RSS, which is mostly determined by the distance.

GNU Radio runs its own thread. Since the RSS data is a continuous feed, like several other physical layer parameters, it becomes necessary to use multithreading to implement the convenience needed for the programmer. The `RSSReader` class implements a thread that starts in the constructor of the class and stops in the destructor.

3) Advanced Directional SDR Algorithms: To illustrate the convenience of our testbed for directional SDR methods, we implement a naive algorithm for detecting AoA. The algorithm initializes the beam direction to zero degrees and rotates the beam until observes a reduction in the RSS. The following function implements this simple technique:

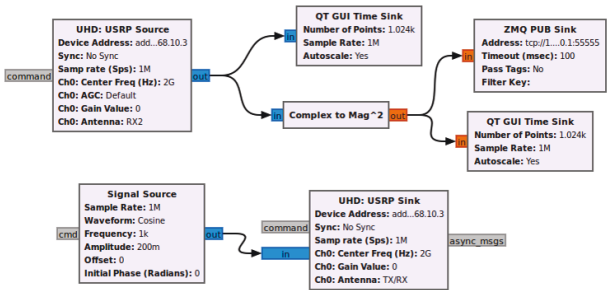


Fig. 5: GNU Radio with ZMQ socket blocks

```
def detectAoA():
 myRSSReader = RSSReader("tcp://127.0.0.1:50001")
 currentAngle = 60; # initialize the starting angle
 bestRSS = 0;
 setAngle(currentAngle);
 currentRSS = myRSSReader.readRSS();
 while (bestRSS <= currentRSS):
    bestRSS = currentRSS;
    currentAngle = currentAngle + 1;
    setAngle(currentAngle);
    currentRSS = myRSSReader.readRSS();
 del myRSSReader;
 return max(currentAngle - 1, 0);
```

The beam will continuously rotate till a peak in the RSS value is observed. The function ends by deleting the `myRSSReader` class object and returning the angle at which the peak RSS was observed. This method assumes that there is only one peak RSS value across the different reception angles, i.e., it assumes there is no multipath or NLOS reception.

IV. Experimental Results

When working with directional wireless channels, aligning the receiver antenna to the best AoA is an important problem in practice. This process involves scanning the reception quality at different angles. To evaluate the impact of `sleepTimer` on the performance of our testbed, we implemented a naive AoA detection algorithm (Sec. III-B3) and measured the accuracy of the AoA detection as `sleepTimer` varies. We constructed three scenarios to test the algorithm. In the first scenario (Fig. 6), the Tx antenna positioned and fixed at 90^o at perfect alignment. The Rx antenna rotates starting from 60^o. For the second scenario (Fig. 7), the Tx antenna is misaligned and rotated 135^o to the right, facing the wall. The Rx antenna starts its rotation from 125^o. In both of these scenarios, K band Horn antennas are used to transmit a 26 GHz signal, 2.5ft apart, and placed 1ft away from the wall. In the third scenario (Fig. 8), we modified the second scenario by using Ka band Horn antennas to transmit a 30 GHz signal and placed them closer to each other at 1ft apart, equal to their distance to the wall. 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 Fig. 9(a) of the first scenario, the RSS

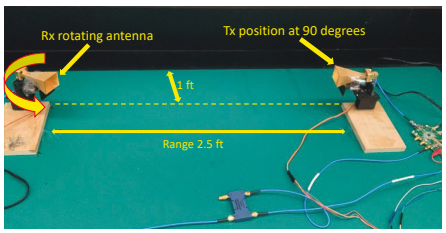


Fig. 6: Aligned: Transmitter at 90^o

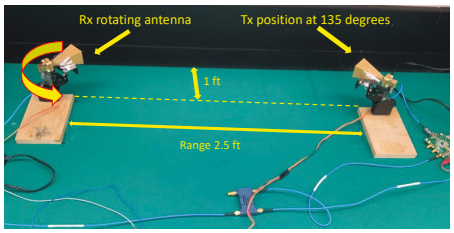


Fig. 7: Misaligned: Transmitter at 135^o

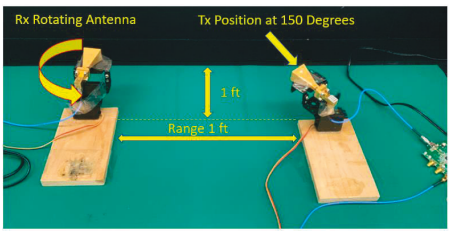
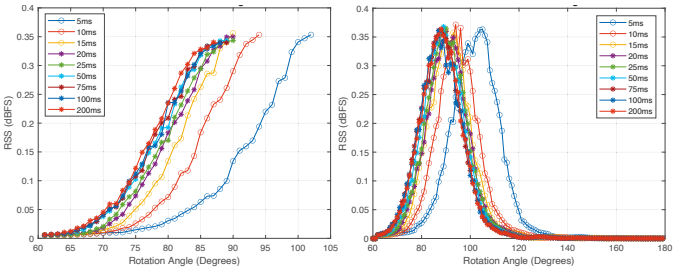
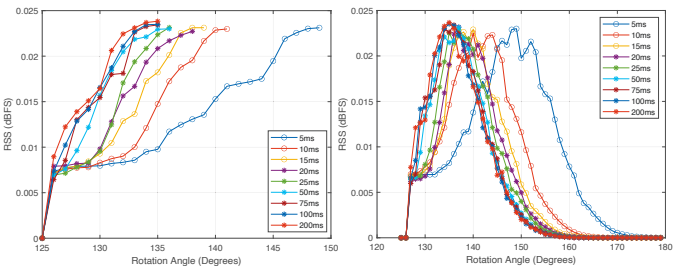


Fig. 8: Misaligned: Transmitter at 150^o



(a) Progress of `detectAoA()` (b) Completed sweep

Fig. 9: Transmitter is aligned at 90^o: RSS vs. the receiver's rotation angle for varying `sleepTimer`

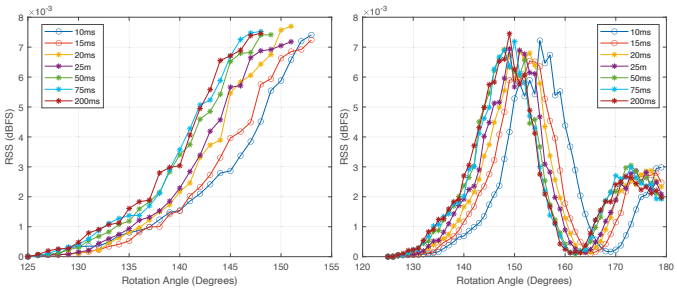


(a) Progress of `detectAoA()` (b) Completed sweep

Fig. 10: Transmitter is misaligned at 135^o: RSS vs. the receiver's rotation angle for varying `sleepTimer`

increases as the Rx antenna is rotated in increments of 1^o. The RSS increases until the beams are aligned at the correct AoA of 90^o. However, 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^o. This inaccuracy becomes unacceptable when `sleepTimer` is lower than 15ms, causing the peak RSS to occur at a rotation angle notably larger than 90^o. This AoA detection error is detailed in Fig. 12. To observe the RSS beyond the desired target AoA, a complete sweep was performed up to 180^o. As expected, in Fig. 9(b), the RSS increases as the Rx antenna approaches to the AoA and slowly decreases as it orients away from the AoA. In the complete sweep, we consistently observe that the accuracy of the measured RSS deteriorates with faster `sleepTimer`.

When there is a misalignment, AoA detection becomes



(a) Progress of `detectAoA()` (b) Completed sweep

Fig. 11: Transmitter is misaligned at 150^o: RSS vs. the receiver's rotation angle for varying `sleepTimer`

more complicated due to multipath signals reflected from the environment. In the misalignment scenarios (Figs. 7 and 8), we observe that the impact of faster `sleepTimer` is stronger as seen in Figs. 10 and 11. Fig. 12 compares the error in AoA detection for the three scenarios, showing similar error behavior against the `sleepTimer`. The completed sweep in the third scenario also shows that there can be multiple humps in the RSS, complicating the AoA detection. As displayed in Figure 11(b), the first/left hump (around 150^o) is the reflection of the main lobe and the second hump (around 175^o) is the reflection of a sidelobe. Our naive AoA detection algorithm inspects the RSS measurements at each angle and successfully finds the correct AoA. However, this is merely because it started its sweep from 125^o.

Obviously, more sophisticated algorithmic designs are necessary to increase the likelihood of finding the correct AoA in minimal amount of time. One simple improvement would be to continue scanning until another hump in the RSS values is observed. Further, instead of inspecting every angle one by one, the algorithm can look at the rate of change in the RSS values and accordingly increase or decrease its speed of scanning the reception angles, e.g., go with large increments if the RSS is not increasing fast enough and reduce the increments on the reception angle as the rate of increase on RSS reduces. Even further, the receiver radio system can learn the behavior of the directional channel over time and guide its AoA detection based on time of day. For example, the number of humps in the RSS values make be more or fewer during different times of the day. Implementation of these advanced AoA detection mechanisms will be very convenient in our testbed by simply using Python libraries.

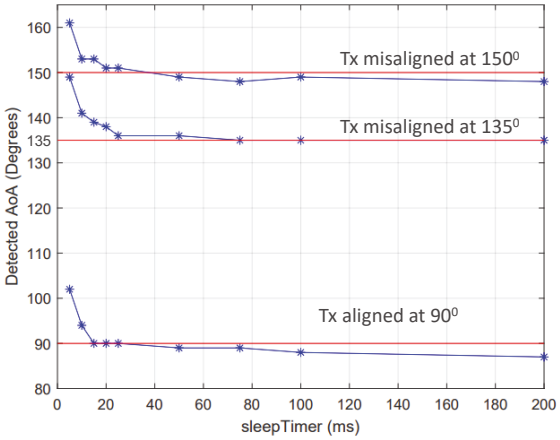


Fig. 12: `sleepTimer` vs. detected AoA

One of the key disadvantages of our system demonstration is the sleep time delay, `sleepTimer`, involved with steering the angle using servo motors, along with the horn antennas. Our beam steering is performed in the order of milliseconds. More expensive systems use electronically steerable patch/phased array antennas in order to alter beam direction. Testbeds such as COSMOS can achieve beam steering in the order of microseconds [25]. Our frontend is flexible and capable of using a variety of antenna types, including patch/phased arrays, which can be used for future research work. Further, unlike other testbeds, our proposed architecture grants the user the ability to customize their software via open source code. This allows the user to control the flow of data, such as RSS and AoA, using multi-threading via Python. This makes the system more convenient, especially for those that are new to GNU Radio.

V. Summary and Future Work

To respond to increasing need for directional wireless methods, we designed a mmWave testbed that allows manipulation of beam direction from a high-level programming language, Python. The testbed utilizes a programmable RF chain that up/down-converts mmWave signals for integration with USRPs, features the open-source GNU Radio for legacy communication and signal processing modules for programming the USRP, and offers software constructs allowing convenient programming of beam direction while enabling access to advanced programming libraries of Python. The testbed successfully runs GNU Radio with other parallel threads for streaming data and implementing beam-steering of mmWave horn antennas. We demonstrated the testbed's efficacy by implementing a simple AoA detection algorithm.

Several exciting directions of directional SDR future work are possible with the capability to program directionality of mmWave beams. Importing machine learning as well as other advanced algorithmic libraries of Python will allow testing of sophisticated solutions to well-known problems in directional wireless (e.g., AoA detection, tracking and localization, and beam alignment) as well as emerging problems of interest such as compressed sensing of the channel in presence of intelligent surfaces. The testbed itself has sizable room for improvement. For example, utilizing another USRP and beam-steering capability on the transmitter side will enable more advanced directional SDR methods, placing the transmit and receive units on portable platforms will enable mobile experimentation, and integrating more powerful FPGAs for datapath processing can enhance the effective bandwidth while maintaining programming flexibility.

VI. Demo: AoA Detection with Q-Learning

Machine Learning (ML) can be subdivided into three categories, supervised, unsupervised, and reinforcement learning. Numerous ML methods have been used to determine AoA. Methods such as K-clustering and Support Vector Machine (SVM) [26], [27] are, respectively, unsupervised and supervised approaches that have been used to determine AoA. Other algorithms, such as Multiple Signal Classification (MUSIC) [28], include a subspace approach which decomposes the received signal into two sub-spaces that include both the signal and noise subspace. The subspace information can then be used to determine the AoA. The MUSIC algorithm does not perform well in the presence of multi-path signals [29]. Therefore, this algorithm may not be ideal to use when signals are exposed to noisy environments commonly experienced in mmWave channels.

To illustrate the usability of our test-bed in real-time mmWave beam-steering, we implement a model-free reinforcement learning algorithm, Q-learning, to detect the AoA on the RX side. Q-learning is widely used for a number of applications, such as video gaming and localizing drones [30].

A. Mapping AoA Detection Problem to Q-Learning

Q-learning is characterized by parameters such as agent, action, reward, and state space. The agent performs an action within the environment. Every action taken results in either a positive or negative reward. Further, once an action is taken, the agent moves onto the next state. Fig. 13 presents the Q-learning diagram of our experimental setup. In our setup, the agent resides at the Rx horn antenna and can perform two actions, either turn left or right by one degree resolution. The reward is defined as the difference between the current and previous RSS values, which rewards the actions turning the antenna towards the AoA and penalizes otherwise. The state space can be any value from 0 to 180 degrees with one degree resolution. Therefore, our state-action table, also called Q-table, can be represented by 180 rows of angular states and 2 columns of actions. As the learning progresses, the Q-table is populated according to the Bellman equation [30]

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

where s_t is the current state, s_{t+1} is the next state chosen by the agent, r_t is the difference between the RSS at times t and $t-1$ (i.e., $RSS_t - RSS_{t-1}$), $\alpha \in (0, 1)$ is the learning rate, and γ is the discount factor.

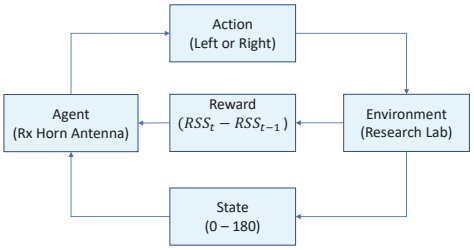


Fig. 13: Q-Learning Block Diagram

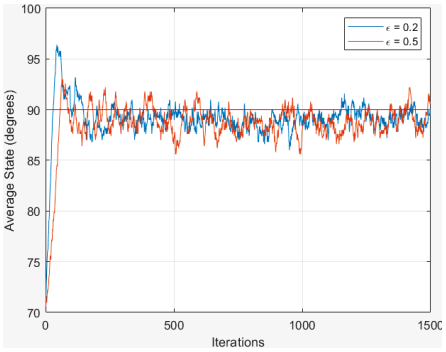


Fig. 14: States of Q-Learning

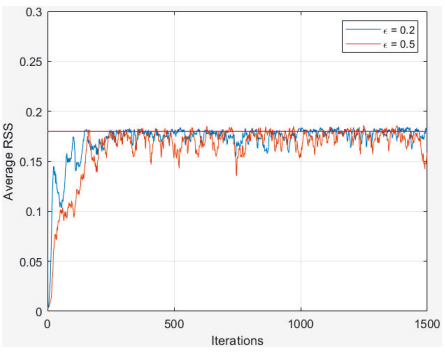


Fig. 15: Average RSS

Q-learning uses a greediness parameter, $\epsilon \in (0, 1)$, in deciding which actions to take. Higher ϵ increases the likelihood that the Q-learning agent chooses a random action while lower ϵ makes the agent exploit the Q-table by selecting an action that results in the current maximum Q value. At every iteration, the agent will take an action based off this greediness and update the Q-table using (1), and learn from the environment. After sufficient iterations, the Bellman equation in (1) guarantees that the learning converges to a solution.

B. Initial Results

For our demo presentation, the transmitter is fixed at 90 degrees pointing towards the receiver horn antenna. α is initialized to 1 and linearly decays with step size 10^{-5}. γ is set to 0.98. Q-learning continues for 1,500 iterations for five total runs. The `sleepTimer` is set to 200ms.

Figs. 14 and 15, respectively, show the average state of the agent (i.e., the steering angle of the Rx antenna) and the average RSS measured by the agent as the Q-learning continues when ϵ is set to 0.2 and 0.5. In all the experiments the agent's initial state is set to 70 degrees. As seen in Fig. 14, the agent starts exploring the environment, and turns left within the first few iterations. It further explores Rx angles beyond 90 degrees and learns that the reward deteriorates as it gets away from the correct solution. Then, it turns right beginning to converge to the desired AoA of 90 degrees. The variances of the average state (in Fig. 14) visited by the agent are 3.748 and 6.276 for $\epsilon = 0.2$ and $\epsilon = 0.5$, respectively. This verifies that the agent is acting in a more exploratory manner when ϵ is higher.

These initial results show that our testbed can implement Python ML methods to perform mmWave beamsteering in real-time. The results above are for a simple case where the transmitter is pointing directly towards the receiver. The demo will illustrate the concept for various scenarios including multipath setups. We will also present convergence analysis of the Q-learning based AoA detection for mmWave channels.

ACKNOWLEDGMENT

This work was supported in part by U.S. National Science Foundation award 2115215.

REFERENCES

[1] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6G wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, Sep. 2019.

[2] 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 Networks*, vol. 21, no. 8, pp. 2657–2676, Nov. 2015.

[3] T. Chen, M. Kohli, T. Dai, A. D. Estigarribia, D. Chizhik, J. Du, R. Feick, R. A. Valenzuela, and G. Zussman, "28 GHz channel measurements in the COSMOS testbed deployment area," in *Proceedings of ACM mmNets Workshop*, 2019, p. 39–44.

[4] 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 Transactions on Signal Processing*, vol. 62, no. 8, pp. 2111–2126, April 2014.

[5] J. Zhang, X. Zhang, P. Kulkarni, and P. Ramanathan, "OpenMili: A 60 GHz software radio platform with a reconfigurable phased-array antenna," in *Proceedings of ACM MOBICOM*, 2016, pp. 162–175.

[6] S. K. Saha, Y. Ghasempour, M. K. Haider, T. Siddiqui, P. De Melo, N. Somanchi, L. Zakrajsek, A. Singh, R. Shyamsunder, O. Torres, D. Uvaydov, J. M. Jornet, E. Knightly, D. Koutsonikolas, D. Pados, Z. Sun, and N. Thawdar, "X60: a programmable testbed for wideband 60 GHz WLANs with phased arrays," *Computer Communications*, vol. 133, pp. 77–88, 2019.

[7] H. Wu, T. Wang, Z. Yuan, C. Peng, Z. Li, Z. Tan, B. Ding, X. Li, Y. Li, J. Liu, and S. Lu, "The Tick programmable low-latency SDR system," in *Proceedings of ACM MOBICOM*, 2017, p. 101–113.

[8] R. Zhao, T. Woodford, T. Wei, K. Qian, and X. Zhang, "M-Cube: a millimeter-wave massive MIMO software radio," in *Proceedings of ACM MOBICOM*, 2020, pp. 1–14.

[9] M. Polese, F. Restuccia, A. Gosain, J. Jornet, S. Bhardwaj, V. Ariyarathna, S. Mandal, K. Zheng, A. Dhananjay, M. Mezzavilla, J. Buckwalter, M. Rodwell, X. Wang, M. Zorzi, A. Madanayake, and T. Melodia, "MillimeTera: toward a large-scale open-source mmWave and terahertz experimental testbed," in *P. of ACM mmNets*, 2019, p. 27–32.

[10] 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 *Proceedings of ACM MOBISYS*, 2020, p. 1–13.

[11] 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. of ACM mmNets Workshop*, 2020.

[12] A. Quadri, H. Zeng, and Y. T. Hou, "A real-time mmWave communication testbed with phase noise cancellation," in *Proceedings of IEEE INFOCOM Workshops*, 2019, pp. 455–460.

[13] H. Hassanieh, O. Abari, M. Rodriguez, M. Abdelghany, D. Katabi, and P. Indyk, "Fast millimeter wave beam alignment," in *Proceedings of ACM SIGCOMM*, 2018, pp. 432–445.

[14] V. Va, H. Vikalo, and R. W. Heath, "Beam tracking for mobile millimeter wave communication systems," in *Proc. of IEEE GlobalSIP*, 2016, pp. 743–747.

[15] C. Zhang, D. Guo, and P. Fan, "Tracking angles of departure and arrival in a mobile millimeter wave channel," in *Proceedings of IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[16] "UBX 10-6000 MHz Rx/TX (40 MHz, N series and X series)," Ettus Research, a National Instruments Brand. [Online]. Available: https://www.ettus.com/all-products/ubx40
[17] O. Abari, H. Hassanieh, M. Rodreguiz, and D. Katabi, "A millimeter wave software defined radio platform with phased arrays," in *Proceedings of ACM MOBICOM*, 2016, p. 419–420.
[18] "ADMV1013," Analog Devices. [Online]. Available: https://www.analog.com/en/products/admv1013.html
[19] "ADMV1014," Analog Devices. [Online]. Available: https://www.analog.com/en/products/admv1014.html
[20] "GNU Radio - the free and open source radio ecosystem," GNU Radio. [Online]. Available: https://www.gnuradio.org
[21] "About USRP bandwidths and sampling rates." [Online]. Available: https://kb.ettus.com/About_USRP_Bandwidths_and_Sampling_Rates
[22] E. M. Fennelly, *The Utilization of Software Defined Radios for Adaptive, Phased Array Antenna Systems*. The University of Vermont and State Agricultural College, 2020.
[23] 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 *Proc. of IEEE/MTT-S Int. Microwave Symposium*, 2018, pp. 1323–1326.
[24] E. Fennelly and J. Frolik, "Phase measurement and correction for software defined radio systems," in *Proc. of IEEE WAMICON*, 2021, pp. 1–5.
[25] T. Chen, P. Maddala, P. Skrimponis, J. Kolodziejski, X. Gu, A. Paidimarri, S. Rangan, G. Zussman, and I. Seskar, "Programmable and open-access millimeter-wave radios in the PAWR COSMOS testbed," in *Proceedings of ACM WinTech*, 2022, pp. 1–8.
[26] M. Pastorino and A. Randazzo, "A smart antenna system for direction of arrival estimation based on a support vector regression," *IEEE Tran. on Antennas and Propagation*, vol. 53, no. 7, pp. 2161–2168, 2005.
[27] M. Roshanaei and M. Maleki, "Dynamic-knn: A novel locating method in wlan based on angle of arrival," in *2009 IEEE Symposium on Industrial Electronics & Applications*, vol. 2, 2009, pp. 722–726.
[28] A. Barabell, "Improving the resolution performance of eigenstructure-based direction-finding algorithms," in *Proc. of IEEE Int. Conference on Acoustics, Speech, and Signal Processing*, vol. 8, 1983, pp. 336–339.
[29] Z. Dai, Y. He, T. Vu, N. Trigoni, and A. Markham, "Deepaoanet: Learning angle of arrival from software defined radios with deep neural networks," 2021. [Online]. Available: https://arxiv.org/abs/2112.00695
[30] M. M. U. Chowdhury, F. Erden, and I. Guvenc, "Rss-based q-learning for indoor uav navigation," 11 2019, pp. 121–126.