A Computer Vision Based Testbed for 77 GHz mmWave Radar Sensors

Onur Toker
Dept. of ECE
Florida Polytechnic University
Lakeland, FL 33805
otoker@floridapoly.edu

Suleiman Alsweiss
Dept. of ECE
Florida Polytechnic University
Lakeland, FL 33805
salsweiss@floridapoly.edu

Muhammad Abid
Dept. of Comp. Sci.
Florida Polytechnic University
Lakeland, FL 33805
mabid@floridapoly.edu

***Abstract*—In this paper, our main objective is to develop a tool (testbed) that allows real-time testing and visualization of mmWave radar algorithms using a high-level programming language. This tool is capable of capturing data from the radar module without any loss of packets and efficiently display results as heatmap images of target distance and velocity. This new testbed is especially tailored for hardware-in-the-loop (HIL) type of experiments involved in Advanced Drivers Assistance Systems (ADAS) and Autonomous Vehicles (AV) among others. To validate the efficacy of the developed testbed, Texas Instruments (TI) AWR1642 module was used. Results show that the test bed described hereafter is successful and the proposed hardware and software configuration will allow more advanced image processing and computer vision techniques to be used for a variety of applications. Several examples are shown with Python being the main programming language for control and data acquisition and openCV for visualization of radar images.**

***Index Terms*—Automotive radars, Real-time testing and visualization, Hardware in the loop.**

I. Introduction

Because of the increasing demand for autonomous driving and driver assistance features in consumer vehicles, research on Advanced Drivers Assistance Systems (ADAS), and Autonomous Vehicles (AV) are increasing at unprecedented rates. There are several sensor technologies which are considered to have key importance in ADAS and AV systems [1], namely: Camera based sensors, lidars, and mmWave radars. Camera based sensors are quite popular because of their lower cost and the way they perceive the environment in a way similar to human eyes. However, they are susceptible to different weather and lighting conditions,and they do require more processing power compared to other alternatives. On the other hand, lidars are gaining popularity because of their advantages in 3D mapping, but they do require relatively expensive manufacturing steps.The third key technology is mmWave radars [2], specifically those operating at the 77-81 GHz band. Costs associated with this sensor technology are comparable to camera based sensors, in addition to having various advantages including ease of obstacle detection, target range and velocity estimation, direction estimation, the ability to work under day/night conditions, etc.

978-1-7281-6861-6/20/$31.00 ©2020 IEEE

The focus of this paper is mmWave radar based sensors. In this work, we will be using Texas Instruments (TI) AWR family of automotive radars [3], and FPGA based low voltage differential signaling (LVDS) to Gigabit Ethernet companion boards [4]. This work is a continuation of the authors' earlier research [5], [6]. As a related work, we would like to cite [7]–[10] where a different 77 GHz radar setup is used.

A fundamental difficulty for research on applications of mmWave radars in ADAS and AV systems is the lack of low-cost real-time testbeds suitable for Hardware in the Loop (HIL) experiments. TI's radar development kits and the companion FPGA boards are reasonably priced, all have multi-input multi-output (MIMO) architecture, and comes with basic software to record raw ADC data. However, like other manufacturers, TI's existing setup is not suitable for HIL experiments, and it is only usable for offline analysis of the recorded data. This makes the use of TI's C/C++ toolchain for rapid prototyping of new ideas, and HIL tests on ADAS and AV systems is not a practical approach.

Thus, the aim of this paper is to develop a real-time mmWave radar testbed which is (1) suitable for HIL experiments on ADAS and AV systems, (2) supports a high-level programming language for rapid prototyping, (3) supports for real-time visualization of radar images for faster and/or more effective performance analysis, and finally (4) supports commonly used computer vision and artificial intelligence AI libraries.

The remainder of this paper is organized as follows. A review of the basics of frequency modulated continuous wave (FMCW) radar systems is presented in Section 2. In Section 3, we present some background information on TI's mmWave radars, in Section 4, we discuss the design and implementation of a high performance multi-threaded Ethernet capture and parser subsystem, in Section 5, we explain the programming model of the developed real-time testbed, in Section 6, we present a sample openCV based design to process radar images for real-time moving object tracking, and finally in Section 7, we make some concluding remarks.

II. Basics of FMCW Radar Processing

FMCW radars are considered the primary radar architecture for ADAS and AV applications [11]. When equipped

with multiple transmitters and receivers, beam forming and direction estimation are possible which makes FMCW radars even more suitable for the problem at hand [12]. This section will provide an overview on how FMCW radar chirp and beat (intermediate frequency, IF) signals can be used for range and velocity estimation of targets using the concepts of frames and pulsed-Doppler signal processing.

Typically, an FMCW radar system has a voltage controlled oscillator (VCO), for which the frequency of oscillation is controlled by an external input voltage signal $q(t) \in [0, q_M]$. Without loss of generality, we will assume that $q_M = 1$ to simplify the analysis. The VCO output frequency depends on the input voltage in a nonlinear fashion. However, in most designs, VCO is operated in the linear-like frequency-to-voltage region to avoid applying any non-linearity corrections [13]. Therefore, the VCO output frequency can be modeled as:

$$f(t) = f_0 + Bq(t)$$

where f_0 is the VCO output frequency at the minimum input voltage , and B is the VCO bandwidth. In FMCW radar systems, VCOs are usually driven by triangular or saw tooth signal to create a linear chirp, and the period of the triangular signal is called the chirp duration (T_c). It is possible to use both positive and negative chirps to drive the VCO,

$$q_+(t) = t/T_c, \quad \text{or} \quad q_-(t) = (T_c - t)/T_c,$$

and regardless of the sign, the slope of the chirp signal (S) can be calculated as: $S = B/T_c$.

In an FMCW radar system, the VCO output is amplified, and fed to a transmitter antenna(s). The transmitted signal is called the chirp signal ($S_T(t)$), and can be expressed as:

$$S_T(t) = A_T(t) \cos\left(2\pi \int^t f(\tau)d\tau + \theta_T\right),$$

where $A_T(t)$ is the amplitude of the VCO output signal, and θ_T is its phase. The received signal (echo) from a target at distance $d(t)$ will then be:

$$S_R(t) = A_R(t) \cos\left(2\pi \int^{t-t_r} f(\tau)d\tau + \theta_R\right)$$

where $t_r = 2d(t)/c$ is the round-trip time with c being the speed of light. Once an echo is received, the transmitted and received signals are first multiplied in a mixer, then low pass filtered (LPF), and the resulting signal is called the beat signal, $s(t)$, which can be expressed as:

$$s(t) \approx A(t) \cos\left(2\pi f(t) t_r + \theta\right).$$

For a positive chirp, the beat signal frequency will be

$$f_{b+} = \frac{2Sd_0}{c} + \frac{2f_0 v_0}{c},$$

where d_0 and v_0 are the target distance and velocity at $t = 0$ respectively. In principle, it is possible to use beat signals resulting from a positive and negative chirp, and estimate both target distance and velocity. However, for moving targets, there is a more effective procedure using frames, which will be discussed later in this section.

Typically, the beat signal is digitized after being LPF for further processing. If a beat signal is sampled at frequency f_s, then its frequency can be estimated using the fast Fourier transform (FFT) algorithm with accuracy f_s/N where $N = f_s T_c$ and represents the number of the beat signal samples obtained during the chirp duration. Therefore, the resolution in target distance estimation (δd) can be estimated as:

$$\delta d = \frac{c}{2S} \frac{f_s}{f_s T_c} = \frac{c}{2B},$$

which indicates that VCOs with higher bandwidth will have better distance (range) resolution. Moreover, the maximum range of the FMCW radar (d_{max}) is limited by the sampling frequency f_s of the analog to digital converter (ADC) used to digitize the beat signal, and can be expressed as:

$$d_{max} = \frac{f_s c}{2S},$$

meaning that sampling the beat signal with a higher sampling frequency actually improves the maximum distance where a target can be detected.

If the target is moving, beat signals corresponding to two consecutive chirps will have a phase difference equal to

$$\delta\phi = 2\pi f_0 \frac{2v_0 T_c}{c} = \frac{4\pi v_0 T_c}{\lambda},$$

where λ is the wavelength of the radar signal at frequency f_0.

For unambiguous velocity estimation, the absolute value of the induced phase difference must be less than π, therefore the maximum possible velocity is limited by

$$v_{max} = \frac{\lambda}{4T_c}.$$

In FMCW radar systems, a frame is defined as M chirps combined together, and the corresponding beat signal data is organized as a complex matrix, F, with each row containing a single beat signal corresponding to a single chirp. For moving target detection, 2D FFT of the frame matrix, F, can be used. If we have M chirp signals with time separation of T_c, the resolution in velocity will be

$$\delta v = \frac{\lambda}{2MT_c},$$

namely the velocity resolution will be proportional with the number of chirps, M.

In summary, if we have a single beat signal, one-dimensional (1D) FFT and the corresponding peaks can be used for target detection. While if we have a frame, then two-dimensional (2D) FFT of the corresponding matrix F can be used for range and velocity estimation, and can be presented either as a gray scale image or it can be color coded for better visualization. These color coded images are called radar range-velocity heatmaps, or simply radar images, and are quite effective for moving target detection.

III. TI mmWAVE RADARS

In this section, we will review the architecture of TI's AWR1642 FMCW radar. At the chip level, there is a single VCO operating at 20 GHz followed by a 4x frequency multiplier [14]. This chip can drive two transmit antennas with separate power amplifiers and phase shifters. Furthermore, this chip is designed to support four receive antennas each having its own low noise amplifier, mixer, low pass filter, and 16 bits ADC. Each receive channel has its own 90^o phase shifters too, and both in-phase and quadrature components are digitized. In other words, from a signal processing perspective, we have a complex beat signal which is sampled at a very high sampling rate. The AWR1642 module has an ARM Cortex-R4F processor, which is called the master subsystem (MSS), and a C674x DSP accelerator called the DSP subsystem (DSS).

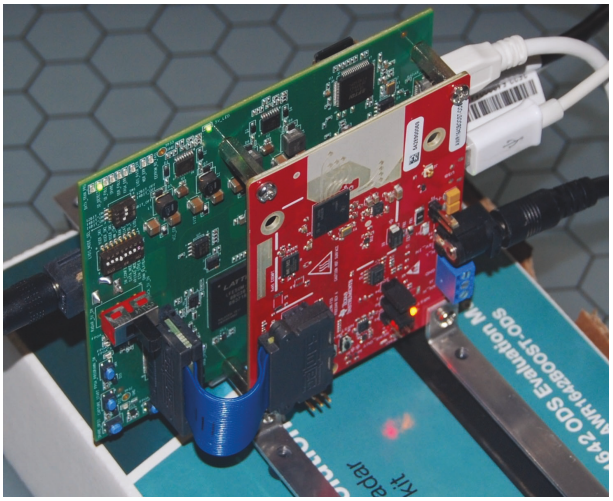


Fig. 1. 77GHz AWR1642 radar (red board) connected to a DCA1000 FPGA board (green board) via an LVDS cable (blue flat cable).

In Fig. 1, our AWR1642 radar is shown connected to a DCA1000 FPGA board over an LVDS cable. The raw ADC data is sent to the FPGA board over the LVDS cable, and then from the FPGA board to the host PC over a Gigabit Ethernet cable for further processing. TI's default setup is designed to record this raw ADC data for offline processing. While this was very helpful, it was not convenient, and the authors developed an alternative testbed for processing the data in real-time. The developed testbed supports real-time visualization for better performance analysis, and hardware in the loop experiments for testing high level algorithms on a real radar sensor used in a real experiment.

The AWR1642 radar has various parameters which can be configured using the mmWave Studio GUI, see Fig. 2. AWR1642 radar's maximum bandwidth is 4 GHz, each chirp is generated in a 160 μs time window (default value), and the ADC sampling frequency is 10 MHz. During each chirp, a total of $N = 256$ samples (default value) are acquired from each receive channel. Chirps are repeated $M = 128$ times

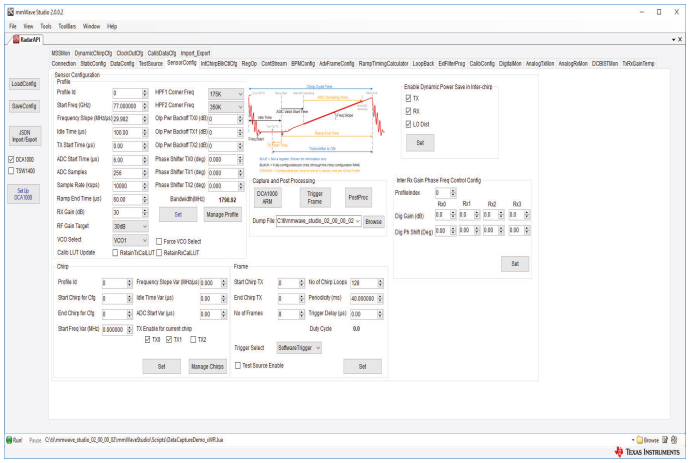


Fig. 2. mmWave Studio configuration GUI.

(default value) over a 40 ms time window, which is also called the frame duration. A frame consists of multiple chirps (M chirps) followed by a blank period at the end during which no chirp signal is generated and no data is captured (See Fig. 3).

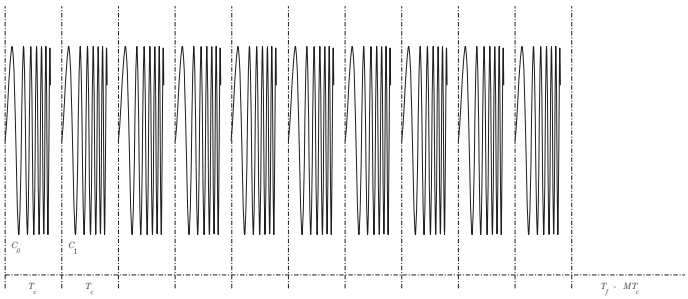


Fig. 3. An AWR1642 frame consists of M chirps.

IV. MULTI-THREADED ETHERNET CAPTURE AND PARSER IMPLEMENTATION

In this section, the design and implementation of the developed multi-threaded Ethernet capture and parser program is outlined. As mentioned in the previous section, the raw ADC data captured by the AWR1642 is sent to the DCA1000 FPGA board over an LVDS cable (See Fig. 1). The FPGA board gets the raw ADC data and sends it to the host PC over a Gigabit Ethernet connection as user datagram protocol (UDP) packets using the format shown in Fig. 4. The TI documentation [4] has detailed information about all possible formats that are supported, but the testbed developed in this work is based on the format given in Fig. 4.

In the developed Python code, UDP data is read as 1466 byte "chunk" using the `recvfrom()` method of the `socket` class, which returns a `bytes` object for which the first 10 bytes are sequence number and byte count, and the rest are actual raw ADC data. Sequence number starts from 1, and byte count is the number of raw ADC data bytes sent before this packet.

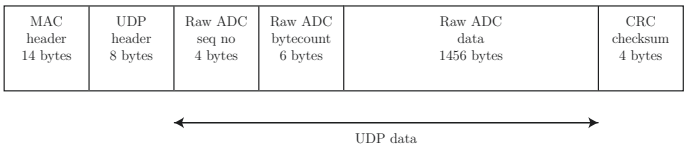


Fig. 4. AWR1642 Ethernet packet format.

The first feasibility tests completed by authors was to test whether this raw UDP data can be streamed to a numpy array without any parsing or post-processing. The following Python code was used and no dropped packets were observed.

```
frm = np.empty((250,360,1466), dtype=np.uint8)
for fc in range(250):
    seqno = None
    for i in range(360):
        packet, _ = sock.recvfrom(1466)
        pno = int.from_bytes(packet[:4], byteorder='little')
        frm[fc,i,:] = np.frombuffer(packet, dtype=np.uint8)
        if (seqno == None):
            seqno = pno
        else:
            if (pno > seqno + 1):
                print("IOError")
            seqno = pno
```

The raw ADC data streamed from the FPGA board to the host computer has a specific structure as explained in [4], and presented in Fig. 5. Each of these boxes in Fig. 5 represents a 16-bits ADC value, the first index is the antenna number from 0 to 3, the second index is either I or Q, i.e. real or imaginary part, and the third index is the sample number.

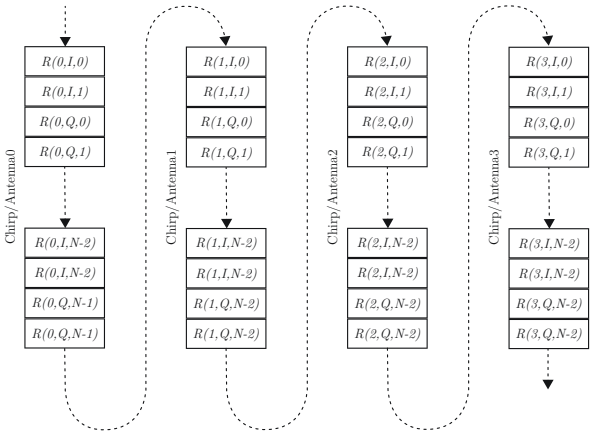


Fig. 5. DCA1000 stream format.

For a single chirp, the total data size can be calculated as:

$$\text{chirp data size} = 4 \times 256 \times 2 \times 2 = 4096 \text{ bytes}$$

representing data from a total of 4 antennas, 256 samples/chirp, 2 bytes/sample, and $\times 2$ to account for the complex data type (i.e. real and imaginary parts). A single chirp data will be split over $4096/1456 \approx 2.8$ Ethernet frames (or UDP packets), which necessitates parsing and data re-organization. In a single frame of duration 40 ms, a total of 128 chirps are generated for each antenna. Therefore,

$$\text{frame data size} = 128 \times 4096 = 512 \text{ KB}$$

and this will be split over ≈ 360 Ethernet frames (or UDP packets). Thus, the overall raw data rate will be 100 Mbps (excluding all overhead data), or more than 130 Mbps including all header info. This level of data rate is not possible by using a 100 Mbps Ethernet connection, hence Gigabit Ethernet is required. Furthermore, the average number of Ethernet frames (UDP packets) sent per second is ≈ 9002, so for real-time performance, a carefully designed parsing and data re-organization implementation is necessary.

In an earlier version of this work [6], the design was capable of capturing, parsing, and processing approximately 10 chirps per second. Basically, chirp data and/or 1D FFT results were captured at a rate of 10 updates per second as shown in see Fig. 6, and was possible to display live target distance as a sliding graph window as demonstrated in the YouTube video [15]. However, this design was single threaded, had nested loops, and the code organization was very similar to the MATLAB parser presented in [4].

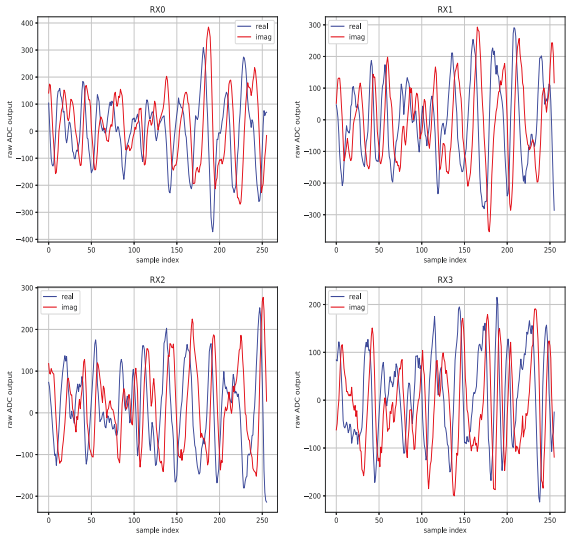


Fig. 6. Previous design presented in [6]. Raw ADC data from all receive antennas with real and imaginary parts can be displayed with ≈ 10 updates/sec.

To process 25 frames per second, with each frame having 128 chirps per second, the authors implemented a multi-threaded design shown in Fig.7.

There are a total of four threads, three of which are doing parsing and data reorganization, and the last one is the final consumer thread doing openCV based number crunching. Threads are separated by thread-safe first-in first-out (FIFO) buffers FX0, FX1, and FX2. The first thread simply reads the received UDP packets, checks sequence numbers for

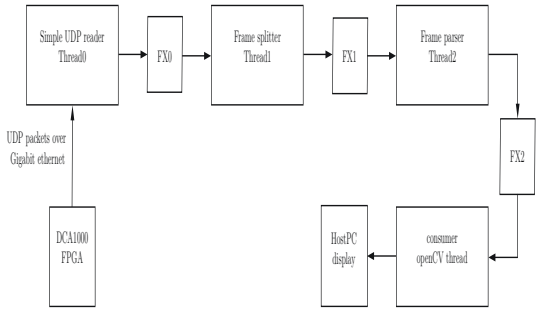


Fig. 7. Multi-threaded design for real-time performance. FX0, FX1, and FX2 are FIFO buffers.

continuity, and writes the received packets to the FIFO buffer FX0 without further processing. The second thread serves as a frame splitter where the beginning and the end of each frame data are found and passed to the frame parser thread. Finally, the frame parser thread takes care of the data reorganization and generates the final numpy arrays of dimensions $4 \times M \times N$ for each received data frame (recall that there are 4 receive antennas, M chirps in a frame, and N samples in a chirp). Each generated numpy array for the received frame data is written to the final FIFO buffer, FX2. Each thread code is highly optimized for real-time performance, but in the event of a packet loss, they are designed to skip the data for the current frame, and continue with the next frame. Therefore, even if a small portion of a frame data is lost because of a dropped UDP packet, the whole radar frame is discarded. This design choice was made simply to optimize for real-time performance. Multiple experiments were conducted and almost no packet loss was detected. However, having multiple applications running on the host computer can increase the likelihood of packet loss.

Finally, depending on the nature of the application, the consumer thread will get the parsed frame data as a numpy matrix ready to be used in openCV and/or other tools. The consumer thread will be similar to a typical openCV loop, but it will start with `fifo2.get()` to get the radar frame data from the FIFO buffer FX2. Typically, in an openCV loop, `cap.read()` reads the image from a video capture device as a numpy array, whereas here we have `fifo2.get()` to get the frame data from the FIFO buffer. This will allow users to take advantage of all available openCV functions as if they are developing a real-time openCV application using a video capture device.

Fig.8 depicts an unprocessed radar image in a live openCV window. This is basically the 2D FFT of the frame data after FFT shifting and absolute value operations. Although the original image is in gray scale, it is shown in PARULA colormap for better visualization (warmer colors indicate stronger echo). The bright regions correspond to reflections from different targets, in particular bright regions along the center horizontal region correspond to reflections from stationary objects. Bright regions above the center horizontal region correspond to objects approaching the radar.

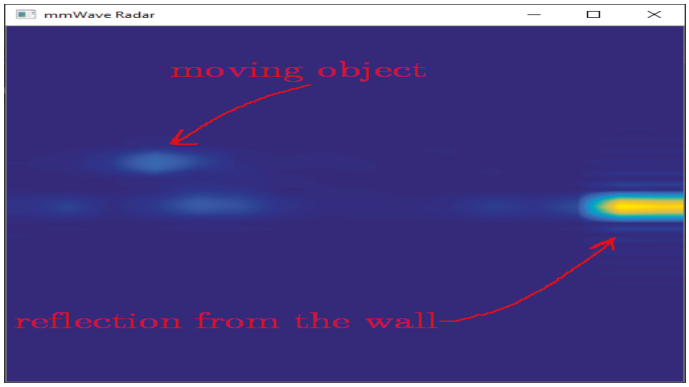


Fig. 8. Unprocessed radar image in a live openCV window (Horizontal axis is distance, and vertical axis is velocity).

V. Programming Model Comparison

In the previous version of this testbed [6], the Python library `matplotlib` was used for visualization of 1D data. The programming model introduced in [6] was based on using an "animate" function which is called iteratively about 10 times per second. At the beginning of the "animate" function, chirp data is loaded as a 1D numpy vector, then all DSP operations are done using numpy and scipy methods on this numpy array before saving results to a `matplotlib.pyplot.figure` object. The following code fragment show the programming model used in [6]

```
def animate(i):
    try:
        tir.clear_buffer()
        f = tir.capture_frame()
        s = np.matmul(np.ones((1,tir.rx)), f)
        s = s.reshape((-1,))
        S = fft(s, M)
        S = np.abs(S[:kmax])
        S = S / M
    except:
        S = np.zeros(kmax)
    line.set_ydata(S)
    return [line]

anim = FuncAnimation(fig, animate, init_func=init,
                                 interval=100, blit=True)
plt.show()
```

In this earlier design, although the name of the method is `capture_frame()`, it was only able to capture up to 8 consecutive chirps with approximately 10 updates per second. In other words, a more appropriate name would be `capture_chirp()`. However, using the multi-threaded optimized design described in the previous section, it is now possible to capture all 25 frames per second, with each frame consisting of 128 chirps, each chirp 256 complex samples per antenna, and display openCV processed 2D FFT images in real-time with no dropped data. A typical openCV thread is shown below.

```
def openCV_thread():
    [some initialization]
    while True:
        _, r_data = fifo2.get()
```

```
        im = r_data[0,:,:]            # use the 1st antenna
        im = window_matrix * im
        fm = np.fft.fftshift(np.fft.fft2(im, (2 * M, N)))
        fm = np.abs(fm)

        [openCV operations on gray scale image fm]
        [normalize and optional resize in openCV]

        im_gray = np.array(fm * 255, dtype=np.uint8)
        im_color = cv2.applyColorMap(im_gray,
                                     cv2.COLORMAP_PARULA)

        cv2.imshow('mmWave Radar', im_color)

        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
```

A. Background subtraction

In Fig. 9, a processed radar image is shown in a live openCV window. Compared to the image shown in Fig. 8, an adaptive background subtraction algorithm is applied.

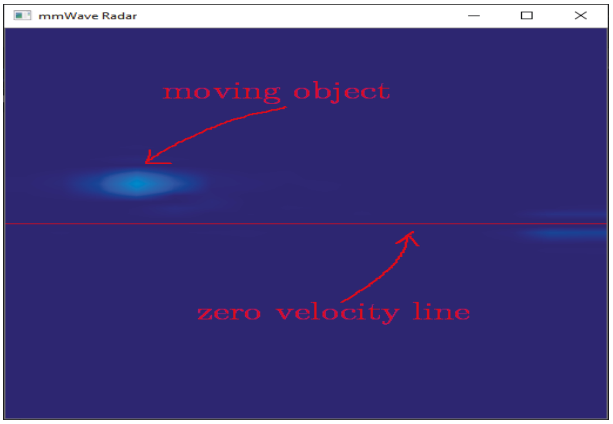


Fig. 9. Radar image after background subtraction shown in a live openCV window (Horizontal axis is distance, and vertical axis is velocity).

Basically, after 2D FFT of the frame data is obtained, the background for the k^{th} image is estimated as:

$$b_k[i,j] = 0.95 b_{k-1}[i,j] + 0.05 f_k[i,j]$$

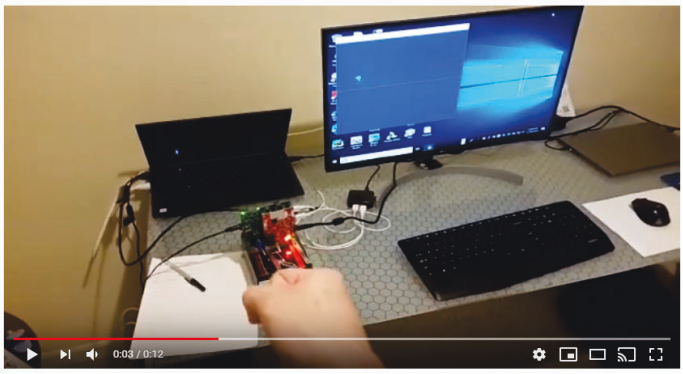
where i, j are row and column indices, f_k is the original gray scale radar image, and b_k is the estimated background. What is displayed in Fig. 9 is g_k which is defined as

$$g_k[i,j] = \max(f_k[i,j] - 0.90 b_k[i,j], 0),$$

followed by some constant gain compensation.

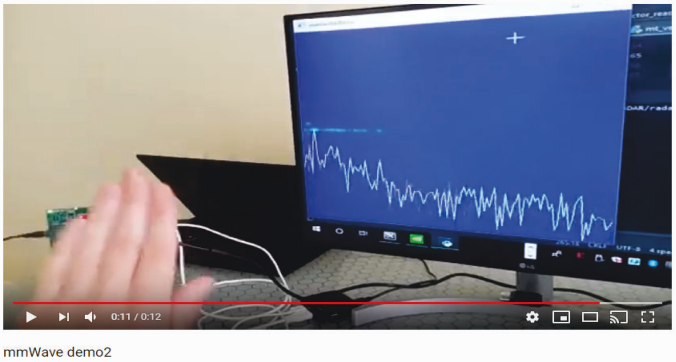
B. Demo videos

To test the efficacy of the new testbed, it was used in three different experiments summarized in Figs. 10-12 with YouTube links shown in the figures captions. In Fig. 10, a different adaptive background subtraction and adaptive normalization is used, and in Fig. 11, both 2D FFT of the frame data, and 1D FFT of the chirp data are shown. In Fig. 12, background is not subtracted but gamma correction is applied to emphasize lower brightness pixels. All of these are real-time experiments without using any recorded data, and they clearly demonstrate the effectiveness of the developed system for real-time processing and visualization.



moving object detection with background subtraction (live demo)

Fig. 10. https://www.youtube.com/watch?v=2kEqh4AuMGo



2D FFT of frame and 1D FFT of chirp data (live demo)

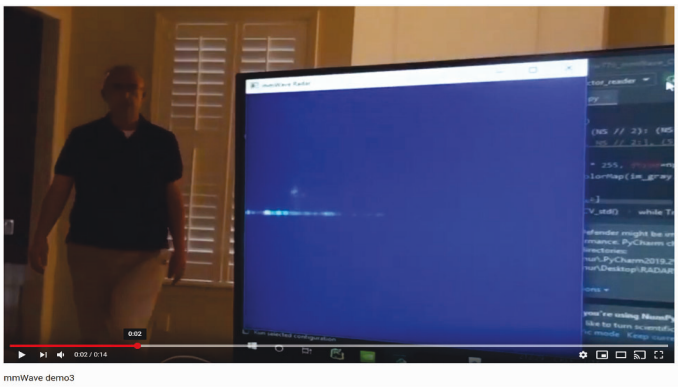
Fig. 11. https://www.youtube.com/watch?v=MDlA2KCt7BA

VI. Moving Object Tracking

In this section, we use the developed testbed to test a specific object tracking algorithm in real-time. Object tracking in the radar image corresponds to tracking the velocity and the range of the object in a real test environment. That's why object tracking in radar images is of extreme importance, especially in ADAS and AV applications.

First, the image background was estimated as described in the previous section, and 90% of it was subtracted from the actual gray scale image, which is the 2D FFT of the frame data after applying FFT shift and taking the absolute value. afterwards, the mean and the variance of the background subtracted image were computed using the openCV method `cv2.moments()`,

$$m_x = \frac{\sum x g[x,y]}{\sum g[x,y]}, \quad m_y = \frac{\sum y g[x,y]}{\sum g[x,y]}$$



Walking pedestrian (live demo)

Fig. 12. https://www.youtube.com/watch?v=jNuBgl6KMCs

and

$$\sigma_x^2 = \frac{\sum x^2 g[x,y]}{\sum g[x,y]} - m_x^2, \quad \sigma_y^2 = \frac{\sum y^2 g[x,y]}{\sum g[x,y]} - m_y^2$$

where all pixels of the background subtracted image g were included in the summations. The center of the bounding box is (m_x, m_y) and the corners are $(m_x \pm \sigma_x/2, m_y \pm \sigma_y/2)$. A sample frame is shown in Fig. 13 with the computed bounding box shown in yellow. The center red line is the zero velocity line.

The m_x and m_y values can be used to estimate target distance, and target velocity by using simple proportionality relations. The spread in the radar image, which can be measured by (σ_x, σ_y), is related to several factors, including, but not limited to, the number of samples in a chirp (256 samples), number of chirps in a frame (128), windowing function used (Hanning window), openCV resizing operations, and target's motion during a single frame duration of 25 ms.

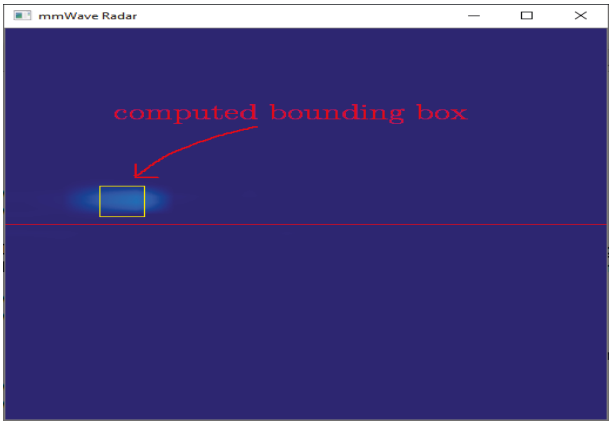


Fig. 13. Moving object tracking after background subtraction. Computed bounding box shown in a live openCV window.

VII. Conclusion

In this work, we presented the design and implementation of a real-time mmWave radar testbed with real-time openCV processing and visualization. The robustness of the developed testbed makes it suitable for hardware in the loop (HIL) configuration of ADAS and AV research. **Using the developed testbed, TI radar echo signals can be dealt with as a camera with 128×256 resolution and 25 fps. This simplified programming model is one of the main contributions of this work.** Real-time openCV based processing of this resolution and frame rate in a Python environment is not computationally extensive, and looks quite promising for future experimental research. Other future research directions may include more complex image and video processing, and the use of multiple views/directions by using complex weighted sums of frame data from multiple antennas.

Acknowledgment

This work is supported Florida Polytechnic University, Advanced Mobility Institute (AMI), and National Science Foundation under Grant No. CNS-1919855. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] R. Kala, *On-Road Intelligent Vehicles*. Butterworth-Heinemann, 2016.
[2] K. Ramasubramanian, K. Ramaiah, and A. Aginskiy, "Moving from legacy 24 GHz to state-of-the-art 77 GHz radar," White Paper, Texas Instruments, Oct. 2017.
[3] "AWR1642 evaluation module (AWR1642BOOST) single-chip mmwave sensing solution," User Guide, Texas Instruments, Apr. 2018.
[4] "DCA1000EVM data capture card," User Guide, Texas Instruments, May 2019.
[5] M. Brinkmann, O. Toker, and S. Alsweiss, "Design of an FPGA/SoC hardware accelerator for MIT coffee can radar systems," in *IEEE SouthEastCon 2019*, Huntsville, AL, Apr. 2019.
[6] O. Toker and B. Kuhn, "A python based testbed for real-time testing and visualization using TI's 77 GHz automotive radars," in *2019 IEEE Vehicular Networking Conference (VNC)*, Los Angeles, CA, Dec. 2019.
[7] C. D. Ozkaptan, E. Ekici, and O. Altintas, "A software-defined OFDM radar for joint automotive radar and communication systems," in *2019 IEEE Vehicular Networking Conference (VNC)*, Los Angeles, CA, Dec. 2019.
[8] M. Ash, M. Ritchie, and K. Chetty, "On the application of digital moving target indication techniques to short-range FMCW radar data," *IEEE Sensors Journal*, vol. 18, no. 10, pp. 4167–4175, 2018.
[9] J. Park, Y. Hong, H. Lee, C. Jang, G. Yun, H. Lee, and J. Yook, "Noncontact RF vital sign sensor for continuous monitoring of driver status," *IEEE Trans. on Boimedical Circuits and Systems*, vol. 13, no. 3, pp. 493–502, 2019.
[10] M. Aalizadeh, G. Shaker, J. C. M. Almeida, P. P. Morita, and S. Safavi-Naeini, "Remote monitoring of human vital signs using mm-wave FMCW radar," *IEEE Access*, vol. 7, pp. 54 958–54 968, 2019.
[11] S. M. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques," *IEEE Signal Proc. Mag.*, vol. 34, pp. 22–35, 2017.
[12] C. Iovescu and S. Rao, "The fundamentals of millimeter wave sensors," Texas Instruments, 2017.
[13] O. Toker and M. Brinkmann, "A novel nonlinearity correction algorithm for FMCW radar systems for optimal range accuracy and improved multitarget detection capability," *MDPI Electronics*, vol. 8(11), no. 1290, pp. 1–13, 2019.
[14] "AWR1642 single-chip 77- and 79-GHz FMCW radar sensor," Data Sheet, Texas Instruments, Apr. 2018.
[15] "Live distance measurement demo using TI's AWR1642 and the ti77radar python library," YouTube video, https://youtu.be/T0YE5dZwlBY, 2019.