THE LEADING JOURNAL OF ENGINEERING, APPLIED SCIENCE, AND TECHNOLOGY

Print ISSN: 2157-8052
Online ISSN: 1930-6628

www.ijme.us

www.iajc.org

INTERNATIONAL JOURNAL OF MODERN ENGINEERING

ABOUT IJME:

- IJME was established in 2000 and is the first and official flagship journal of the International Association of Journal and Conferences (IAJC).
- IJME is a high-quality, independent journal steered by a distinguished board of directors and supported by an international review board representing many well-known universities, colleges and corporations in the U.S. and abroad.
- IJME has an impact factor of **3.00**, placing it among the top 100 engineering journals worldwide, and is the #1 visited engineering journal website (according to the National Science Digital Library).

OTHER IAJC JOURNALS:

- The International Journal of Engineering Research and Innovation (IJERI)
  For more information visit www.ijeri.org
- The Technology Interface International Journal (TIIJ).
  For more information visit www.tiij.org

IJME SUBMISSIONS:

- Manuscripts should be sent electronically to the manuscript editor, Dr. Philip Weinsier, at philipw@bgsu.edu.

  For submission guidelines visit
  www.ijme.us/submissions

TO JOIN THE REVIEW BOARD:

- Contact the chair of the International Review Board, Dr. Philip Weinsier, at philipw@bgsu.edu.

  For more information visit
  www.ijme.us/ijme_editorial.htm

INDEXING ORGANIZATIONS:

- IJME is currently indexed by 22 agencies. For a complete listing, please visit us at www.ijme.us.

Contact us:

Mark Rajai, Ph.D.

Editor-in-Chief
California State University-Northridge
College of Engineering and Computer Science
Room: JD 4510
Northridge, CA 91330
Office: (818) 677-5003
Email: mrajai@csun.edu

www.tiij.org

www.ijeri.org

INTERNATIONAL JOURNAL OF MODERN ENGINEERING

The INTERNATIONAL JOURNAL OF MODERN ENGINEERING (IJME) is an independent, not-for-profit publication, which aims to provide the engineering community with a resource and forum for scholarly expression and reflection.

IJME is published twice annually (fall and spring issues) and includes peer-reviewed research articles, editorials, and commentary that contribute to our understanding of the issues, problems, and research associated with engineering and related fields. The journal encourages the submission of manuscripts from private, public, and academic sectors. The views expressed are those of the authors and do not necessarily reflect the opinions of the IJME editors.

EDITORIAL OFFICE:

Mark Rajai, Ph.D.
Editor-in-Chief
Office: (818) 677-2167
Email: ijmeeditor@iajc.org
Dept. of Manufacturing Systems Engineering & Management
California State University-Northridge
18111Nordhoff Street
Northridge, CA 91330-8332

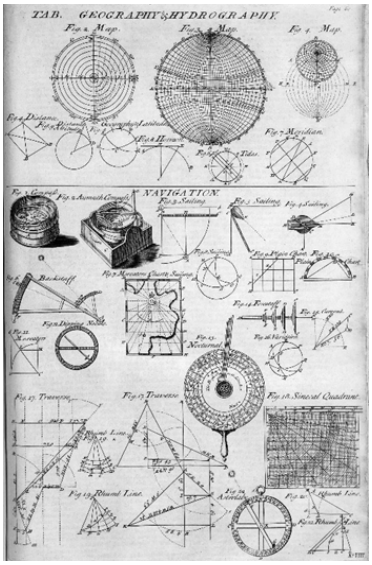
THE INTERNATIONAL JOURNAL OF MODERN ENGINEERING EDITORS

TABLE OF CONTENTS

In This Issue (p.5)
Marine Vessel Navigation Systems

Philip Weinsier, IJME Manuscript Editor

- The *SS Andrea Doria* and *MS Stockholm* collision (1956)
- The *RMS Titanic* allision (1912)
- The *SS Edmund Fitzgerald*—sunk either by a storm or a collision with the *SS Arthur M. Anderson* (1958)
- The *Exxon Valdez* oil spill allision (1989)
- The *Costa Concordia* allision (2012)



Remember any of these maritime disasters? Every year, many people are injured or killed when maritime vessels collide (both vessels are in motion) or are involved in an allision (a moving vessel impacts a stationary vessel or a fixed object). According to the Insurance Information Institute (www.iii.org), shipping losses around the world over the last decade included over 700 vessels—the bulk of these by S. China, Indochina, Indonesia, and the Philippines. But shipping losses as a category doesn't give us the "big picture." Marine vessel incidents between 2014 and 2021 included 11,000+ cargo ships, 5000+ passenger ships, 4000+ fishing vessels, 2200+ service ships, and 660+ "others" (www.SAFETY4SEA.com). According to the United States Coast Guard, the top contributing factors to these collisions are operator inattention and/or inexperience and an improper lookout.

At the risk of stating the obvious, anything that can reduce the number of incidents and losses should be evaluated and considered. Case in point: ARPA (Automatic RADAR Plotting Aid systems), required by the larger commercial vessels to automatically detect potential collisions and provide audio/visual alerts. The problem, though, is cost; owners of smaller vessels—such as those used by average recreational boaters—simply are not able to afford such systems. It is possible, however, to implement a low-cost ARPA-like system using open source software in conjunction with consumer-grade, commercially available RADAR systems.

A typical shipboard ARPA/radar system.

In the 20th century, the Decca Navigator System launched its first hyperbolic radio navigation system that allowed ships and aircraft to determine their position by using radio signals from a dedicated system of static radio transmitters. LORAN (LOng RAnge Navigation) was another hyperbolic radio navigation system developed in the United States during World War II and offered an improved range, up to 1500 miles. More modern methods of ship collision avoidance include the following.

- ECDIS (Electronic Chart Display and Information System)
  A GPS-based system used for route planning and automatic ETA computations as well as pinpointing tricky or congested routes.
- AIS (Automatic Identification System)
  Transmits real-time ship data in order to detect vessels in poor weather.
- RADAR (RAdio Detection And Ranging)
  Detects obstacles and vessels in low visibility, such as nighttime navigation in crowded harbors.
- COLREGS (International Regulations for Preventing Collisions at Sea)
  COLREGS-mandated lights are specific configurations of navigation and signaling lights for indicating a vessel's position, status, and intent to nearby ships.

In our featured article (p.5), the authors evaluate aspects associated with such an implementation and discuss the use of the OpenCV computer vision library to automatically extract target information from a standard commercial RADAR system and render it on a navigational display for visualization. In addition, the authors present the implementation of target tracking using the multiple hypothesis tracking (MHT) algorithm in conjunction with a Kalman filtering algorithm to predict the position of a detected target via a simulated example.

The goal of their paper is to present a framework that facilitates the implementation of ARPA capabilities in small vessels using consumer-grade sensors and RADAR systems by integrating open source software for target detection and rendering with target-tracking algorithms.

DECCA navigation system.

Editorial Review Board Members

Framework for Implementing Advanced Radar Plotting-Aid Capability for Small Maritime Vessels

Otilia Popescu, Old Dominion University; Jason S. Harris, Old Dominion University; Dimitrie C. Popescu, Old Dominion University

Abstract

Every year, many people are injured or even lose their lives, when small maritime vessels collide with other vessels or fixed objects. According to the United States Coast Guard, the top contributing factors to these collisions are operator inattention and/or inexperience and an improper lookout that could be prevented by using Automatic RADAR Plotting Aid (ARPA) systems. It is worth noting that, while larger commercial vessels are required to have RADAR systems with ARPA capabilities to automatically detect collisions and alert the vessel operator to alter course, the cost of such systems can be prohibitive for small vessels used by average recreational boaters. Nevertheless, it is possible to implement a low-cost ARPA- like system by using open source software in conjunction with consumer-grade RADAR systems that are commercially available for use on small vessels. In this study, the authors evaluated aspects associated with such an implementation and, in this paper, discuss the use of the OpenCV computer vision library to automatically extract target information from a standard commercial RADAR system and render it on a navigational display for visualization. In addition, the authors present the implementation of target tracking using the multiple hypothesis tracking (MHT) algorithm in conjunction with a Kalman filtering algorithm to predict the position of a detected target via a simulated example.

Introduction

Operating small vessels in open waters can be challenging, especially when poor situational awareness hinders their maneuvering around other vessels or fixed obstacles. According to the latest recreational boating statistics released by the United States Coast Guard (USCG, 2022) there were 1085 collisions involving small recreational vessels leading to 39 deaths and 512 injuries. According to the USCG (2022), the top three known primary contributing factors for accidents were operator inattention and/or inexperience and an improper lookout, causing 1453 accidents that resulted in 136 deaths and 791 injuries. While no solution is perfect in reducing accidents on the open seas, technology can be used to augment the skills and capabilities of boat operators. A marine RADAR system is a very important tool for small vessels, when it comes to safety, as it can assist navigators when poor visibility conditions exist, when navigation by sight is not possible. In such scenarios, a RADAR system with ARPA capability can automatically detect nearby obstacles, plot their course, and warn operators of imminent collisions (Bole, Wall, Norris & Dineley, 2005). However, while commercial vessels are required to be equipped with RADAR systems that have ARPA capabilities to provide warnings against potential collisions, most recreational boats and small vessels lack such capabilities. This is due to various factors, such as the cost of the system, the additional weight, and the power requirements if installed systems. It is theorized that, if this technology were more available and affordable, more vessels would be equipped with it and the number of accidents on the water would be reduced.

One of the main challenges in implementing ARPA capabilities on marine RADAR systems for small vessels is being able to provide an accurate platform heading to the ARPA tracking algorithm. This is because the dynamic nature of a vessel operating on water leads to uncertainties regarding the vessel's orientation with respect to the yaw axis; Figure 1 illustrates how this is commonly represented as noise and is combined with the uncertainties due to the RADAR sensor. These uncertainties are much more problematic on smaller vessels, which are not fitted with precision instruments to take vessel bearing measurements, but generally rely on inexpensive consumer-grade sensors to determine their heading. Nevertheless, being able to use the rough information provided by these sensors in conjunction with affordable small marine RADAR systems would allow the implementation of collision detection capability on more vessels, resulting in a safer navigation environment both on shore and on the open seas.

Figure 1. Yaw uncertainty due to the vessels' heading (yellow) adds to the uncertainty of the RADAR sensor (green) to increase the range of variation in the position of a detected vessel.

The goal of this paper is to be a platform for the authors to present a framework that facilitates the implementation of ARPA capabilities in small vessels using consumer-grade sensors and RADAR systems by integrating open source software for target detection and rendering with target-tracking algorithms. The authors were not able to find such a framework in the open literature; thus, providing one will

enable the expansion and improvement of the safety features of small vessels. In this paper, the authors begin with a brief overview of sensors that are available on the consumer market and may be used to provide heading information, augmenting the basic information obtained from simple RADAR systems commonly available on small vessels. This is followed by a discussion outlining how the RADAR system obtains information about targets that are present in the environment and how this information is rendered for target detection and visualization using the OpenCV library. Finally, the authors present a discussion on target tracking, illustrated through simulations.

Heading Sensors for Small Vessels

Small maritime vessels, as defined in this paper, do not exceed 60 feet in length, and include personal watercraft generally owned by individuals for recreational purposes. Such vessels may be easily transported on land, using trailers and commercial off-the-shelf RADAR systems available for these types of comparatively small watercraft, which are relatively inexpensive, costing only a few thousand dollars in the United States. Due to their low cost and size, these RADAR systems usually have no ARPA functionalities for automatically detecting targets and warning operators of potential collisions. However, many modern systems designed for small vessels may include some collision-avoidance functions referred to as Mini ARPA (or MARPA), which are essentially simplified versions of the ARPA used on larger vessels (SIMRAD, 2024). These are available as advanced features usually not included in the price of the RADAR system, and they use information from additional onboard sensors such as a compass or global positioning system (GPS) and require displays with control units (McMillan, 2024).

Nevertheless, implementing ARPA capabilities on commercial off-the-shelf RADAR systems used in small vessels can be accomplished using information that can be obtained from affordable consumer-grade micro-electro-mechanical systems (MEMS) that are already incorporated into most smartphones and can be interfaced with Arduino microcontrollers or Raspberry Pi single-board computers at a fraction of the cost (Abankwa, Johnston, Scott & Cox, 2015). MEMS technology became popular during the 1990s, when it started to be used on a large scale in the automotive industry (Yazdi, Ayazi & Najafi, 1998). Currently, MEMS are considered to be low-cost devices and various types of MEMS sensors are constructed using different techniques (Lin, Xiong, Dai & Xia, 2017), as shown in Figure 2.

Accelerometers are used to measure acceleration and are constructed by using a mass suspended between two small springs. The mass acts as a capacitive plate and, as its relationship changes with respect to a fixed plate, the change in capacitance can be converted to an acceleration value (Rao, Wei, Zhang, Zhang, Hu, Liu & Tu, 2019). Gyroscopes are constructed by combining two accelerometers, such that the accelerometer that is furthest away from the center of rotation will register a larger acceleration, thereby allowing for the measurement of orientation or angular velocity. Torsional magnetometers are formed by building a plate with a coil suspended by a torsion bar, such that when a current is passed through the coil of wire, the Lorentz force will apply a torsion to the plate altering the capacitance with respect to two fixed capacitive plates underneath (Wu, Tian, Ren & You, 2018).

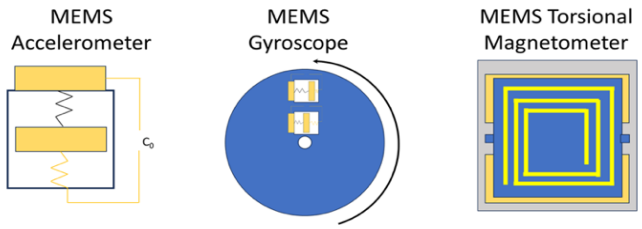


Figure 2. Diagrams of MEMS accelerometers, gyroscopes, and torsional magnetometers.

In recent years, low-cost inertial measurement units (IMUs) that combine accelerometers, gyroscopes, and magnetometers, have been used to determine the orientation of an object (such as a vessel or a gaming controller) with relatively good performance (Patonis, Patias, Tziavos, Rossikopoulos & Margaritis, 2018). It is worth noting that the individual components in an IMU are able to complement each other for accurate determination of the orientation in the three-dimensional space of the object: the accelerometer provides information that can be used to determine the direction in which the sensor platform moves; the magnetometer is able to determine a true measurement with respect to magnetic North; and the gyroscope detects small changes in the orientation of the sensor platform. Table 1 shows update rates and resolutions corresponding to commonly used low-cost IMUs, which are at a sub-second scale (tens or hundreds of Hz update rates with a few thousands of a degree of angular resolution).

Table 1. Magnetometer and gyroscope specifications for popular consumer-grade IMUs.

Device	Magnetometer		Gyroscope	
	Update rate [Hz]	Resolution [mT]	Update rate [Hz]	Resolution [°/s]
MPU-9250	100	0.59	8000	0.076
ICM-20948	100	0.15	9000	0.076
BNO 055	30	0.3	400	0.038

And, while the corresponding data can be used for compensating yaw variations (Yi, Wu, Yue, Zhang, Chen & Wan, 2020), it will have to be down sampled for use in determining the heading of a small vessel, as the vessel's heading changes at sub-second scales, which is considered irrelevant for the sea-keeping ability of the vessel. Sensor fusion algorithms are able to combine accelerometer, magnetometer, and gyroscope data to achieve an absolute heading resolution of approximately one degree RMS (Tomasch & Winer, 2019), with a limiting factor for getting accurate heading data implied by calibration of the sensors. In this direction, the gradient-descent algorithm presented by Madgwick, Harrison, and Vaidyanathan (2011) is an excellent choice for sensor fusion, as it performs on par with alternative proprietary algorithms (Tomasch & Winer, 2019).

Target Detection and Rendering for ARPA Visualization

RADAR systems used on small vessels are monostatic, with the RADAR transmitter and receiver collocated on the vessel, and which usually employ frequency modulated continuous wave (FMCW) sensing to probe the environment for target presence. These RADAR systems are only capable of scanning the environment in the azimuth direction at a constant rate (usually around 48 revolutions/minute) and do not have the ability to focus on specific targets. In FMCW radar systems, the transmitter sends a chirp signal, which is a pulse of duration T whose frequency varies linearly over its duration between frequencies F_1 and F_2. The chirp reflection off the target is delayed by a certain amount of time t_d that depends on the distance to the target that is present in the environment and is given by Equation 1:

$$t_d = \frac{2d}{c} \tag{1}$$

where, d is the distance to the target and c is the speed of light.

At any given instant in time, the offset frequency between the transmitted and reflected chirp signals is given by Equation 2:

$$\Delta f = K t_d = K \frac{2d}{c} \tag{2}$$

where, $K = \frac{|F_2 - F_1|}{T}$ is the chirp frequency slope.

The minimum frequency shift Δf_{min} that the RADAR system can detect corresponds to the smallest frequency offset between the transmitted and reflected chirp signals and defines the resolution bandwidth (RBW) of the RADAR system. This depends on the sampling rate and the size of the fast Fourier transform (FFT) used by the RADAR receiver and determines also its range resolution, which corresponds to the smallest distance between two targets for which they can be separated. The power P_r of the chirp signal reflected by a target that is present in the environment at distance d from the RADAR transmitter is given by the radar equation (Skolnik, 1981) of Equation 3:

$$P_r = \frac{P_t G_t G_r \sigma A_e}{(4\pi)^2 d^4} \tag{3}$$

where, P_t is the power of the transmitted chirp, G_t and G_r are the transmit and receive antenna gains, respectively, A_e is the effective area of the receive antenna, and σ is the radar cross-section (RCS) of the target.

The RCS is also referred to as the radar signature and is a measure of how detectable an object is by radar, and directly affects the power of the reflected chirp signal at the radar receiver, with larger RCS values implying larger reflected powers, indicating that the corresponding objects are more easily detected. The RCS depends on the frequency range of the chirp signal and is determined by measurement for practical targets. For small vessels, the RCS measured at microwave frequencies is on the order of 0.02m^2 for small open boats and 2m^2 for recreational boats and cabin cruisers (Skolnik, 1981), which indicates that, for the latter type of small vessels, the power of the reflected signal at the radar receiver is two times larger than for the former. The range of the RADAR system is implied by Equation 3 and is given by Equation 4, such that, in practical scenarios, it is guaranteed that the radio frequency (RF) power of the chirp reflection P_r is above the sensitivity of the RADAR receiver P_{min} from a target located at distance $d \le d_{max}$, given that the RF power at the RADAR transmitter is at least at the P_t level. Note that the RADAR system can detect both moving and fixed targets (such as anchored ships), and that the RADAR range corresponds to the relative distance between the vessel and the detected target.

$$d_{\max} = \left[\frac{P_t G_t G_r \sigma A_e}{(4\pi)^2 P_{\min}} \right]^{1/4} \tag{4}$$

Visualizing RADAR Information

To sense the environment for the presence of a target, the RADAR system performs scans at given azimuth angles relative to the vessel's direction and returns information in the form of spoke data, which consists of a vector at the given azimuth whose elements represent the power strength of the reflected chirp signal at specific distances away from the vessel. Figure 3 illustrates how spoke data corresponds to quantized values of the RF power of the signal reflected by the target and is rendered for visualization in the form of a heat map image, with warm colors indicating bins corresponding to larger reflected power values and cooler

colors indicating bins in which the reflected power is lower. Specifically, a spoke corresponds to a fraction of a degree in azimuth, and range bins are used to specify distance. The value of the cell is indicated by its color and corresponds to the returned signal strength for that cell, with red denoting the largest power value and blue the lowest power equal to the receiver sensitivity. Occasionally, the received RF power may be split among multiple cells, thereby reducing the overall signal-to-noise (SNR) ratio of the target (Richards, 2005; Oppenheim & Schafer, 1975) and providing only an approximate location of the target in terms of its azimuth angle θ and range d. Modern small RADAR systems are capable of returning several thousand spokes per revolution, resulting in an angular resolution that can be on the order of a fraction of a degree.

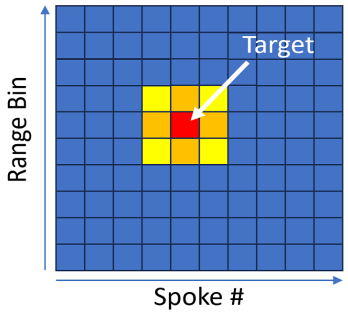


Figure 3. Illustration of RADAR spoke data.

Figure 4 shows that, given the inherent uncertainties of small-vessel RADAR systems mentioned earlier, the spoke number and range bin do not reflect the actual position of the target, but rather place it anywhere inside an ellipse, where the minor axis a implies the uncertainty of the range bins occupied by the target and is determined by the RBW of the RADAR system plus noise, and the major axis of the ellipse b depends on the RADAR beam and is determined by the RBW of the RADAR system plus noise. The major axis of ellipse b depends on the RADAR beam width at the specified range plus the associated noise.

Target Detection Using OpenCV

For target detection, a RADAR scan that consists of 360° of spoke data is used, and its analysis to determine targets is performed by employing standard digital image processing techniques such as blob detection. This can be accomplished using an open source toolkit such as the Open Computer Vision (OpenCV) library, which supports a class called "SimpleBlobDetector" for extracting blobs from an image. The SimpleBlobDetector class uses the "findContours" function, which is an implementation of the algorithm for border detection (Suzuki & Abe, 1985). Once a blob is detected, OpenCV can filter the results to only return blobs that meet area, threshold, circularity, inertia, or convexity requirements. To implement an automatic target extraction method, filtering based on threshold would allow OpenCV to only return blobs with a certain signal strength. It should be noted that the findContours function works only with greyscale images, and the first step in detecting a target is to convert the color RADAR scan image into a greyscale image, followed by inverting the image colors using the OpenCV "bitwise_not" function, since the SimpleBlobDetector function attempts to find blobs that are darker.

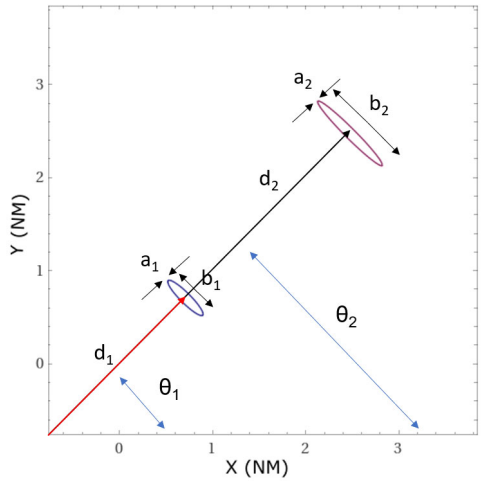
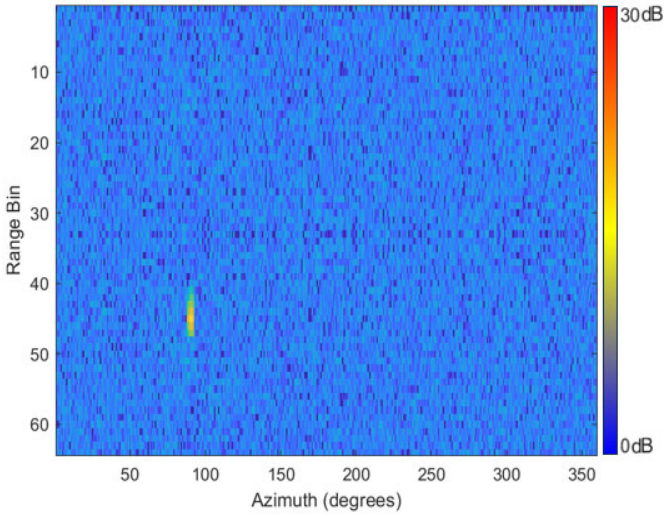


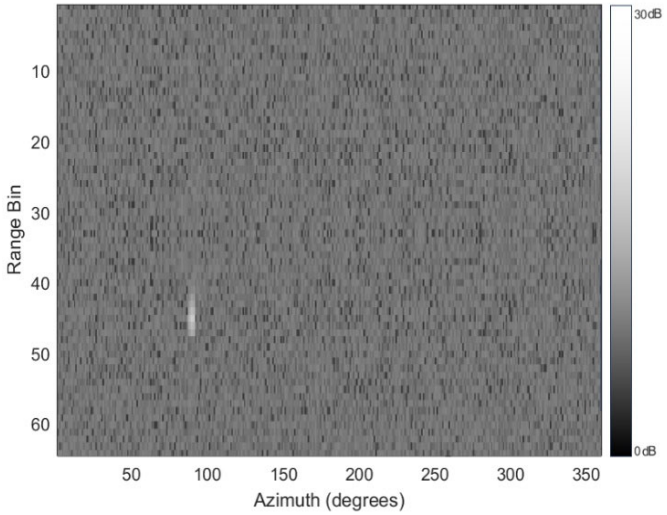
Figure 4. Illustrating uncertainty in target position for two distinct targets located at azimuth $\theta_1 = \theta_2 = 45°$, one with range $d_1 = 1$ nautical mile (NM) and the other with range $d_2 = 3.5$ NM.

The final step is to call the SimpleBlobDetector function, which detects the blobs followed by the "drawKeypoints" function, which allows for an easy rendering of the detected blobs. Figures 5 and 6 illustrate this process on a simulated RADAR scan that includes a singular RADAR return corresponding to a single target with a reflected power of approximately 20 dB above the noise floor. It is important to note that one of the limiting factors of the SimpleBlobDetector function in the OpenCV library is that the greyscale input that it expects is limited to 8 bits. This means that the dynamic range of the function may be considerably more limited than the raw amplitude data returned by the RADAR system, and, as a consequence, targets that are very small or are very far away from the RADAR system may be quantized to an image that is close to the noise level of the system after the conversion to greyscale, resulting in a potential missed detection by the SimpleBlobDetector function.

The OpenCV SimpleBlobDetector function uses two thresholds for detecting blobs, starting at the low threshold value and stepping its way up to the high threshold value, resulting in a series of detected blobs that are then pruned to identify those blobs that are inside of other blobs that correspond to higher detection thresholds. This process is similar to a gradient descent algorithm. For the simulations shown in Figures 5 and 6, the minimum threshold value was set to zero, and the maximum threshold value was set to two standard deviations below the mean of the cell values in the RADAR return.



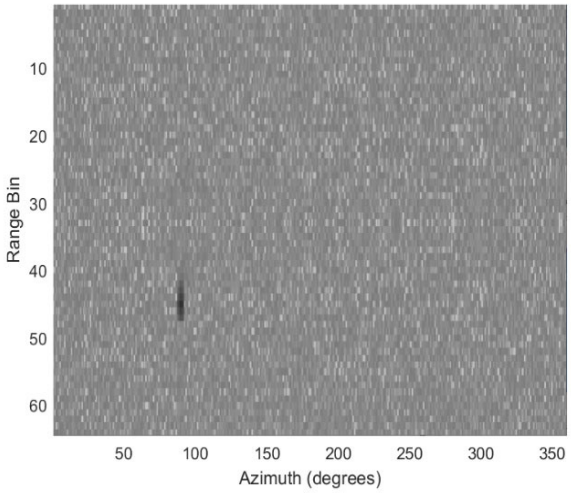
a) Simulated RADAR display with 64 range bins and a resolution of 1° showing a single target.



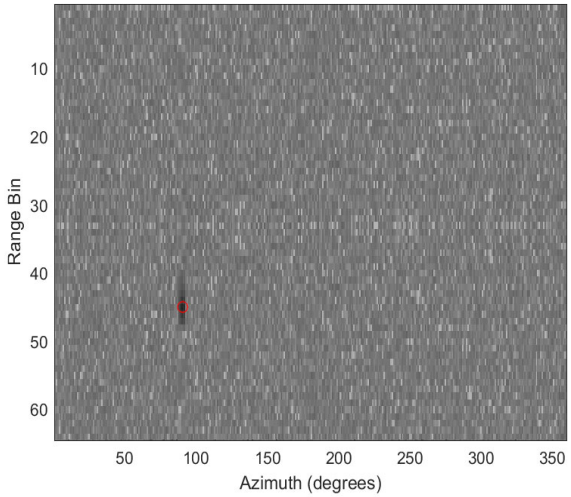
b) Corresponding greyscale OpenCV image.

Figure 5. Simulated RADAR display.

It should be noted that, in the case of large threshold values, the targets that have low reflected powers at the RADAR receiver will be missed, resulting in an increased probability of missed detection, while, for low threshold values, false targets may be detected, resulting in an increased probability of false alarms. In practical settings, the threshold values may be adjusted automatically to ensure a constant false alarm rate (CFAR) (Skolnik, 1981). This uses a cell averaging technique in which the value of the returned power for each cell in the RADAR display is compared against that of the surrounding cells that are situated at a distance of at least two cells to accommodate situations such as those illustrated in Figure 3, where the return power from a target is split across multiple cells. Skolnik (1981) noted that a 1 dB change in the value of the detection threshold can result in a change of three orders of magnitude in the probability of false alarm and, therefore, automatic target detection systems can typically handle less than a 1 dB increase in the noise level (Skolnik, 1981). To determine how effective this technique is for automatic target extraction from RADAR data, a divide-and-conquer algorithm was implemented to simulate RADAR data for a single target with decreasing SNR, and it was determined that this target technique works effectively when the target signal has a SNR of approximately 12 dB or more, supporting ship detection within a range of two nautical miles with an accuracy that approaches 97% (Yulian, Hidayat, Nugroho, Lestari & Prasaja, 2017).



a) Simulated RADAR image of a single target showing OpenCV target detection as a dark blob.



b) Annotation of Figure 6(a) using the "drawKeypoints" function.

Figure 6. Further processing of the simulated RADAR display.

Target Tracking

A RADAR tracking algorithm is an essential component of an ARPA system that takes the coordinates corresponding to multiple observations of detected targets to form a visual track of the target positions in the environment. In this direction, the multiple hypotheses tracking (MHT) algorithm (Reid, 1979; Kim, Li, Ciptadi & Rehg, 2015) has been widely used in radar tracking systems (Blackman & Popoli, 1999). Compared to alternative approaches for target tracking, such as those using nearest neighbors and probabilistic data association (Blackman & Popoli, 1999), the MHT algorithm uses a successive iteration approach that works well in a multitude of scenarios, including track initiation, data clustering with multiple targets, missing measurements and/or false alarms. Blackman and Popoli (1999) found that the MHT algorithm performs significantly better when dealing with dense target environments, which makes it ideal for use in marine applications, since it will work in the dense cluttered target environments of a harbor as well as on the open seas. Figure 7 shows a block diagram of a tracking system based on the MHT algorithm, which details the main steps of the algorithm (Werthmann, 1992).

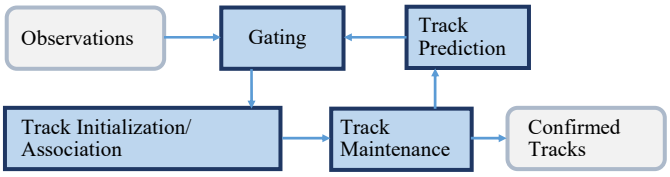


Figure 7. Block Diagram of the tracking algorithm.

The algorithm takes input observations of detected target locations provided by the RADAR system in terms of $\hat{p}$ the target azimuth and range, which are converted to a Cartesian coordinate system for visualization (as discussed earlier). To predict how the location of a detected target is expected to change, the algorithm models the target dynamic using a state-space model that includes the target coordinates x and y along with the corresponding speeds s_x and s_y on the coordinate axes, and uses a Kalman filter for state estimation (also discussed earlier). The observations and the track's predicted locations are processed as part of the gating step, which determines the distance between them and compares it to the gating threshold. The next step consists of track initiation/association in which observations whose distances to the predicted location fall below the threshold, are assigned to existing tracks, while observations whose distances to the predicted location are above the threshold are recorded as possible new tracks.

Following the track initiation/association step, the track maintenance step calculates track scores, eliminating unlikely tracks and confirming surviving ones for track prediction and visual output. Note that in the case of multiple targets that are located close to each other, the SimpleBlobDetector function may end up combining them into a single target, which would result in one or more of the tracks being dropped. However, as the target vessels would maneuver to get away from each other, the SimpleBlobDetector function would eventually be able to distinguish them as separate targets, creating new observations that are not associated with existing tracks. These new observations would prompt the creation of new possible tracks and, over time, the algorithm would build its confidence in the quality of these new tracks, ultimately confirming or disproving them as valid tracks. In order for the RADAR tracking system to work properly it needs to have an accurate location of the target that it is tracking. The signal processing system in the RADAR receiver provides information about the target location in terms of the target azimuth angle and range, which include uncertainties in the tracking platform orientation for range and azimuth, d_{noise} and θ_{noise}, respectively, such that the observations provided to the tracking system are given by Equation 5:

$$\hat{d} = d + d_{noise} \quad \text{and} \quad \hat{\theta} = \theta + \theta_{noise} \tag{5}$$

where, d and θ denote the actual values of the target range and azimuth, respectively.

It is assumed that the uncertainties in the tracking platform orientation for range and azimuth are uncorrelated, that is $E[d_{noise} \cdot \theta_{noise}] = 0$, and that they have zero mean and variances equal to σ^2 and σ^2, respectively. For target location and visualization on a two-dimensional display, the observed range and azimuth values are converted to a position vector $\hat{p}$ containing the Cartesian coordinates of the target, as shown in Equation 6:

$$\hat{p} = \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \begin{bmatrix} \hat{d}\cos(\hat{\theta}) \\ \hat{d}\sin(\hat{\theta}) \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_{noise} \\ y_{noise} \end{bmatrix} \tag{6}$$

Vector p, shown in Equation 7, represents the actual position of the target and includes the Cartesian coordinates of the target:

$$p = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} d\cos(\theta) \\ d\sin(\theta) \end{bmatrix} \tag{7}$$

Vector v, shown in Equation 8, is the vector containing the uncertainties in the target's coordinates, with covariance matrix R shown in Equation 9:

$$v = \begin{bmatrix} x_{noise} \\ y_{noise} \end{bmatrix} \tag{8}$$

$$R = \begin{bmatrix} R_{11} & R_{12} \\ R_{21} & R_{22} \end{bmatrix} \tag{9}$$

The elements of matrix R are given in Equations 10-12, (Longbin, Xiaoquan, Yiyu, Kang & Bar-Shalom, 1998; Duan, Han & Li, 2004):

$$R_{11} = Var\left[x_{noise}\right] = e^{-\sigma_\theta^2/2} d^2 \cos^2\theta + \frac{1}{2}\left(d^2 + \sigma_d^2\right)\left(1 + e^{-2\sigma_\theta^2}\cos 2\theta\right) \quad (10)$$

$$R_{22} = Var\left[y_{noise}\right] = -e^{-\sigma_\theta^2/2} d^2 \sin^2\theta + \frac{1}{2}\left(d^2 + \sigma_d^2\right)\left(1 - e^{-2\sigma_\theta^2}\cos 2\theta\right) \quad (11)$$

$$R_{12} = R_{21} = Cov\left[x_{noise} y_{noise}\right] = e^{-\sigma_\theta^2/2} d^2 \cos\theta \sin\theta + \frac{1}{2}\left(d^2 + \sigma_d^2\right) e^{-2\sigma_\theta^2} \sin 2\theta \quad (12)$$

To predict how the location of a detected target is expected to change and to estimate where the observation for a given track is expected, the target dynamic was modeled using a state-space model, and a state estimation algorithm was used to predict the target position *r*, as shown in Equation 13:

$$r = \begin{bmatrix} x \\ y \\ s_x \\ s_y \end{bmatrix} \quad (13)$$

The state-space model included the actual coordinates of the target *x* and *y* along with the target speed values s_x and s_y on the coordinate axes, which were combined in the target state vector such that the evolution of the target state is given by Equation 14:

$$r\left(k+1\right) = F \cdot r\left(k\right) \quad (14)$$

The state transition matrix *F* is given in Equation 15:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (15)$$

where, Δt is the time interval between state updates.

The initial state vector is shown in Equation 16:

$$r\left(0\right) = \begin{bmatrix} x\left(0\right) \\ y\left(0\right) \\ 0 \\ 0 \end{bmatrix} \quad (16)$$

where, the initial coordinates of the detected target are given by $x(0)$ and $y(0)$ along with zero values for the speeds on the coordinate axes.

The vector containing the target coordinates at time *k* is given by Equation 17:

$$p\left(k\right) = H \cdot r\left(k\right) + v\left(k\right) \quad (17)$$

where, H is the observation matrix given in Equation 18:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (18)$$

and where, *v* is the observation noise in Equation 8 that includes the uncertainties in the target coordinates.

The state estimate for the linear system, described by Equations 14 and 17, is obtained using a Kalman filter and is given by Equations 19-23, (Grewal & Andrews, 2001):

$$\hat{r}_-\left(k\right) = F \cdot \hat{r}_+\left(k-1\right) \quad (19)$$

$$P_-\left(k\right) = F \cdot P_+\left(k-1\right) \cdot F^T \quad (20)$$

$$K\left(k\right) = P_-\left(k\right) \cdot H^T \cdot \left[H \cdot P_-\left(k-1\right) \cdot H^T + R\right]^{-1} \quad (21)$$

$$\hat{r}_+\left(k\right) = \hat{r}_-\left(k\right) + K\left(k\right) \cdot \left[p\left(k\right) - H \cdot \hat{r}_+\left(k\right)\right] \quad (22)$$

$$P_+\left(k\right) = \left[I - K\left(k\right) \cdot H\right] \cdot P_-\left(k\right) \quad (23)$$

where, the "minus" subscript indicates the a priori values of the variables (before the current observations at time instant *k* are used) and the "plus" subscript indicates the a posteriori values of the variables (after the current observations at time instant *k* are used).

Simulations and Discussion

To illustrate target tracking using the approach outlined in the previous section, Figure 8 shows a simulation scenario that was set up with three moving vessels and one vessel serving as the observation vessel, where RADAR information was used to track the other two vessels that represented the moving targets. The trajectories of the three vessels, indicated with lines of different colors in Figure 8, are as follows:

- The red-colored trajectory corresponds to the Observer vessel, which moves along a straight line from South to North at 20 knots.
- The blue-colored trajectory corresponds to target Vessel 1, which moves from West to East, also at 20 knots.

- The yellow-colored trajectory corresponds to target Vessel 2, which moves along a straight line from North to South at 20 knots. It should be noted that the trajectories of the two target vessels were chosen to illustrate two different tracking scenarios.
- Target Vessel 1 is shown on a course that intersects with the Observer vessel, and the bearing from the Observer vessel to Vessel 1 remains the same until they intersect, when it changes as they then move away from each other.
- Target Vessel 2 is shown on a course that passes the Observer vessel on the starboard side, and the bearing from the Observer vessel to Vessel 2 is constantly changing as they approach and then pass each other.

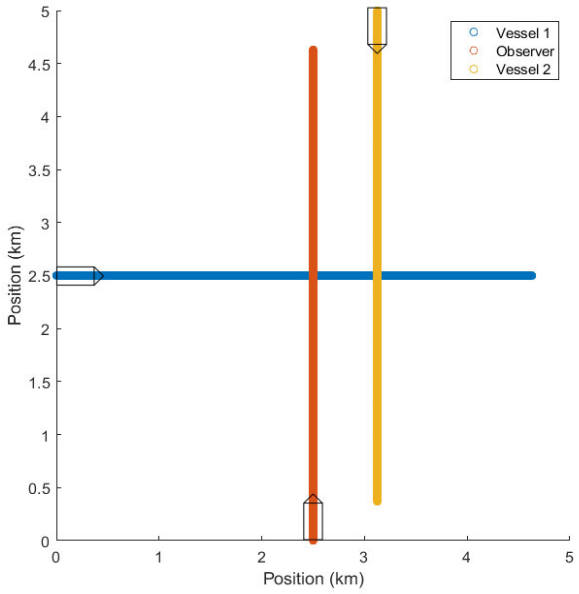
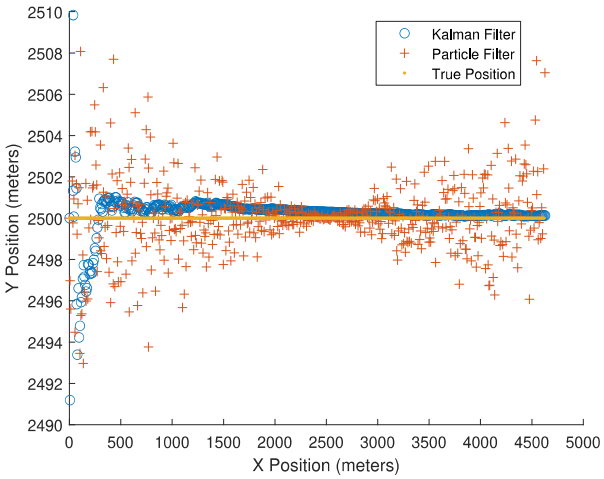


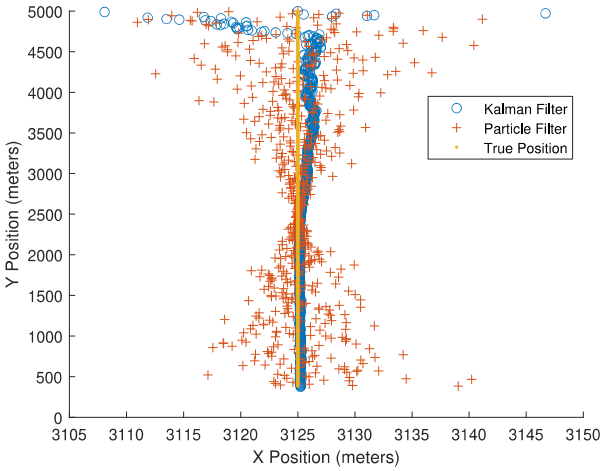
Figure 8. Simulated tracking scenario with one Observer vessel and two target vessels.

For the RADAR system located on the Observer vessel, the bandwidth was set to Δf = 10 MHz, which implies a range bin resolution of approximately 15m, an antenna beam width chosen to be 4°, a transmit antenna array scanning the environment at 48 RPM, and a total simulation time of 7.5 minutes, which resulted in 563 observations of the two targets. For tracking, the implementation was based on the MHT algorithm that used Kalman filtering for track prediction, as discussed in the section on Kalman filtering for track prediction. The gating threshold for the MHT algorithm was set to 30m, and the yaw variances corresponding to the uncertainties in the vessels' headings were chosen to be 0.062° for Vessel 1 and 0.016° for Vessel 2. These values were found using a trial-and-error approach in which the MHT algorithm was simulated multiple times with different values for the yaw variance to determine if it was able to maintain a successful track in 50% of the cases (Harris, 2023).

For comparison, a particle filtering approach for track prediction was also implemented as an alternative to Kalman filtering (Harris, 2023; Ristic, Arulampalam & Gordon, 2004; Elfring, Torta & Van De Molengraft, 2021). Figure 9 shows the simulation results from which it was noted that it took several observations for both the Kalman filter and the particle filter to produce accurate estimates for the positions of the two vessels, with the Kalman filter converging faster than the particle filter to accurate estimates. Moreover, once the Kalman filter converged, the accuracy of the position estimates did not vary significantly, as the position of the Observer vessel changed relative to the two vessels. By contrast, the accuracy of the position estimates produced by the particle filter changed significantly, as the position of the Observer vessel changed relative to the two vessels.



a) Estimated positions for Vessel 1.



b) Estimated positions for Vessel 2.

Figure 9. Estimated positions for Vessels 1 and 2.

It should also be noted from Figure 9 that it was easier for the Observer vessel to acquire a good track for Vessel 1 than for Vessel 2. This was due to the relative position of the Observer vessel as it got closer to the tracked vessel: the bearing from the Observer vessel to Vessel 1 remained the same as the two vessels got closer to each other, whereas the bearing from the Observer vessel to Vessel 2 was changing as the two vessels got closer. Thus, in order to obtain a good track, knowing the orientation of the tracked vessel at all times improves tracking performance. Obtaining this information requires the use of gyroscopes that are capable of producing rapid updates, such as those mentioned in Table 1, that would supplement the position updates observed at slower time scales using consumer-grade GPS.

Conclusions

In this paper, the authors presented a framework for implementing ARPA capabilities on small vessels—for which sophisticated RADAR systems, such as those required for use on large commercial vessels, might prove to be cost prohibitive. Such small-scale, ARPA-capable systems can automatically detect vessels on a collision course and alert the operators to alter their heading. ARPA requires capabilities for target detection and visualization, along with target tracking, to display target positions and predict their paths, which can also be implemented on small vessels using existing affordable technologies. While similar features may be available for small vessels as MARPA to augment the capabilities of their RADAR systems, MARPA is only a basic form of ARPA that requires manual selection of targets before they can be tracked for collision avoidance. Furthermore, MARPA features rely on information from additional onboard sensors and use proprietary algorithms to obtain the vessel heading, and they are usually not included in the base price of the RADAR system, but rather are provided at an additional cost. By contrast, the ARPA implementation presented in the paper can be realized at a fraction of the cost by using consumer-grade MEMS sensors and open source software running on affordable platforms powered by microcontrollers or single-board computers.

Also presented here was how information about targets present in the environment is obtained by using consume-grade RADAR systems, such as the FMCW radars commonly used in small vessels, low-cost MEMS sensors for vessel attitude determination, and the open source computer vision library OpenCV. Specifically, the RF power of the signal reflected by a target is quantized in terms of range and azimuth, and then is rendered for visualization in the form of a heat map image on which the presence of targets can be detected using functions in the OpenCV library. Also discussed was how the coordinates of detected targets are located in a two-dimensional coordinate system for tracking, and how the MHT algorithm with Kalman filtering can be implemented to provide a track of detected targets. The proposed tracking approach was illustrated with a simulation that involved three moving vessels, including one Observer vessel and two detected target vessels, one that was upcoming and another that was crossing from the side. Additional scenarios, such as one in which a vessel is slowly coming up from an angle behind the Observer vessel, will be the object of future work.

References

Abankwa, N. O., Johnston, S. J., Scott, M., & Cox, S. J. (2015, December). Ship motion measurement using an inertial measurement unit. *Proceedings of the 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 375-380. 10.1109/WF-IoT.2015.7389083

Blackman, S., & Popoli, R. (1999). *Design and Analysis of Modern Tracking Systems*. Boston: Artech House.

Bole, A. G., Wall, A. D., Norris, A., & Dineley, W. O. (2005). *Radar and ARPA manual: radar and target tracking for professional mariners, yachtsmen and users of marine radar*. Elsevier.

Duan, Z., Han, C., & Li, X. R. (2004). Comments on "Unbiased Converted Measurements for Tracking." *IEEE Transactions on Aerospace and Electronic Systems*, *40*(4), 1374. DOI: 10. 1109/TAES.2004.1386889

Elfring, J., Torta, E., & Van De Molengraft, R. (2021). Particle filters: A hands-on tutorial. *Sensors*, *21*(2), 438. DOI: 10.3390/s21020438

Grewal, M. S., & Andrews, A. P. (2001). *Kalman Filtering: Theory and Practice Using Matlab*. New York: John Wiley & Sons, Inc.

Harris, J. S. (2023). *Framework for Implementing Advanced Radar Plotting Aid Capability for Small Maritime Vessels Capability for Small Maritime Vessels* (Doctoral dissertation, Old Dominion University). https://digitalcommons.odu.edu/ece_etds/257

Kim, C., Li, F., Ciptadi, A., & Rehg, J. M. (2015). Multiple hypothesis tracking revisited. *Proceedings of the IEEE International Conference on Computer Vision*, 4696-4704). https://doi.org/10.1109/ICCV.2015.533

Lin, Z., Xiong, Y., Dai, H., & Xia, X. (2017, September). An experimental performance evaluation of the orientation accuracy of four nine-axis MEMS motion sensors. *Proceedings of the 5th IEEE International Conference on Enterprise Systems*, 185-189. https://ieeexplore.ieee.org/document/8119388

Longbin, M., Xiaoquan, S., Yiyu, Z., Kang, S. Z., & Bar-Shalom, Y. (1998). Unbiased converted measurements for tracking. *IEEE Transactions on Aerospace and Electronic Systems*, *34*(3), 1023-1027. https://ieeexplore.ieee.org/document/705921

Madgwick, S. O., Harrison, A. J., & Vaidyanathan, R. (2011, June). Estimation of IMU and MARG orientation using a gradient descent algorithm. *Proceedings of the 2011 IEEE International Conference on Rehabilitation Robotics*, 1-7). https://pubmed.ncbi.nlm.nih.gov/22275550/

McMillan, C. (2024). Getting the Best from Your RADAR. https://www.simrad-yachting.com/world-of-simrad/technology/getting-the-best-from-your-radar/

Oppenheim, A. V., & Schafer, R. W. (1975). *Digital Signal Processing*. Englewood Cliffs, New Jersey: Prentice Hall.

Patonis, P., Patias, P., Tziavos, I. N., Rossikopoulos, D., & Margaritis, K. G. (2018). A fusion method for combining low-cost IMU/magnetometer outputs for use in applications on mobile devices. *Sensors*, *18*(8), 2616. https://www.mdpi.com/1424-8220/18/8/2616

Rao, K., Wei, X., Zhang, S., Zhang, M., Hu, C., Liu, H., & Tu, L. C. (2019). A MEMS micro-g capacitive accelerometer based on through-silicon-wafer-etching process. *Micromachines*, *10*(6), 380. https://www.mdpi.com/2072-666X/10/6/380

Reid, D. (1979). An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, *24*(6), 843-854. https://www.scirp.org/reference/referencespapers?referenceid=3887999

Richards, M. A. (2005). *Fundamentals of Radar Signal Processing*. New York, New York: McGraw Hill.

Ristic, B., Arulampalam, S., & Gordon, N. (2004). *Beyond the Kalman Filter: Particle Filters for Tracking Applications*. Boston, MA: Artech House.

SIMRAD. RADAR. https://www.simrad-yachting.com/world-of-simrad/technology/radar/

Skolnik, M.I. (1981). *Introduction to RADAR Systems*. Singapore: McGraw-Hill Book Company.

Suzuki, S., & Abe, K. (1985). Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, *30*(1), 32-46. https://doi.org/10.1016/0734-189X(85)90016-7

Tomasch, G., & Winer, K. (2019). Limits of absolute heading accuracy using inexpensive mems sensors. *Hackaday Journal of What You Don't Know*, *1*(2).

USCG. (2022). Recreational Boating Statistics. https://www.uscgboating.org/library/accident-statistics/Recreational-Boating-Statistics-2022.pdf

Werthmann, J. R. (1992, August). Step-by-step description of a computationally efficient version of multiple hypothesis tracking. *Signal and Data Processing of Small Targets, 1698*, 288-300. https://doi.org/10.1117/12.139379

Wu, L., Tian, Z., Ren, D., & You, Z. (2018). A miniature resonant and torsional magnetometer based on Lorentz force. *Micromachines*, *9*(12), 666. https://doi.org/10.3390/mi9120666

Yazdi, N., Ayazi, F., & Najafi, K. (1998). Micromachined inertial sensors. *Proceedings of the IEEE*, *86*(8), 1640-1659. https://ieeexplore.ieee.org/document/704269

Yi, X., Wu, X., Yue, X., Zhang, L., Chen, Z., & Wan, B. (2020). Ocean surface current inversion with anchored floating high-frequency radar: Yaw compensation. *IEEE Journal of Oceanic Engineering*, *46*(3), 927-939. https://ieeexplore.ieee.org/document/9294035

Yulian, D., Hidayat, R., Nugroho, H. A., Lestari, A. A., & Prasaja, F. (2017, October). Automated ship detection with image enhancement and feature extraction in FMCW marine radars. *Proceedings of the 2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, 58-63. https://ieeexplore.ieee.org/document/8253145

Biographies

OTILIA POPESCU is an associate professor in the Department of Engineering Technology at Old Dominion University. She received her Engineering Diploma in Electrical and Computer Engineering from the Polytechnic Institute of Bucharest in 1991, and PhD in Electrical and Computer Engineering from Rutgers University in 2004. She has experience in wireless communications and networking and is also interested in engineering education, including active learning and undergraduate research. Dr. Popescu may be reached at opopescu@odu.edu

JASON S. HARRIS received his PhD degree in Electrical and Computer Engineering from Old Dominion University in 2024. His interests include radio and antenna systems design, radar, and embedded systems. Dr. Harris is an amateur radio enthusiast and can be reached at kj4iwx@gmail.com

DIMITRIE C. POPESCU is a full professor in the Department of Electrical and Computer Engineering at Old Dominion University. He received his Engineering Diploma in Electrical and Computer Engineering from the Polytechnic Institute of Bucharest in 1991 and PhD in Electrical and Computer Engineering from Rutgers University in 2002. He has experience in wireless communication systems and his interests include software-defined and cognitive radios, transceiver optimization to support quality of service, and signal processing for communications and radar systems. Dr. Popescu may be reached at dpopescu@odu.edu

Design Improvements for Coil Springs in an Automotive Independent Suspension

Diane L. Peters, Kettering University; Yaomin M. Dong, Kettering University; Viraj B. Dave, Kettering University

Abstract

Suspensions are a critical part of vehicle design, as they greatly influence the experience that drivers and passengers have when they are riding in a vehicle. One of the major components of these suspensions is the suspension spring. The springs used in suspensions can take many different forms, including leaf springs, helical springs, and other types. In this current study, the authors used a helical spring, which is the case for many current suspension designs. The focus of the study was on improving the spring, given performance characteristics that were required and subject to a set of constraints to ensure that the spring would not fail in regular operation. FEA analyses were performed on 36 spring designs. At the conclusion of the analyses, the authors found that the suspension could be improved through variations in the spring, in particular by changing the cross section from circular to oval.

Introduction

Suspension systems have a strong impact on both road handling and ride quality of vehicles. Part of their purpose is to ensure that the wheels remain in contact with the road during a vehicle's motion, even when the road is not smooth and has irregularities. Another part of their purpose is to isolate the passengers and cargo from the irregularities in the road, preventing damage to the vehicle and cargo and providing a good experience to the passengers. The type of suspension system used depends in part on the type of vehicle, and the multiple purposes that the suspension serves may result in conflicting constraints. These systems vary both in their mechanical design and in whether they are purely passive or have some active components, which would involve a controller (Peters, Papalambros & Ulsoy, 2013; Tseng & Hrovat, 2015; Riduan, Tamaldin, Sudrajat & Ahmad, 2018). Again, in this current study, the authors considered a purely passive suspension.

In passive suspensions, the performance depends on the type and parameters of the springs and dampers used. In existing suspension designs, three basic types of springs are used: leaf springs (Baviskar, Bhamre & Sarode, 2013; Mahanthi & Murali, 2017), torsion bars (Tavares, Molina, Al Sakka, Dhaens & Ruderman, 2019; Karuppiah, Ganesan, Kasavan & Sambasivam, 2023), and coil springs (Bartolozzi & Frendo, 2011; Lavanya, Rao & Reddy, 2014). At times, air springs, rubber springs, or hydropneumatic springs are also used. (De Melo, Pereira & Morais, 2018; Lijun, Zengliang & Zhuoping, 2010; Joo, 1991).

Suspension designs can be categorized based on their overall goals and principles. One such type of suspension is the set of anti-dive/anti-squat designs. These suspensions typically have differences between the front and rear suspensions, and are designed to counteract the tendency of the vehicle for the front to dive under braking and the rear to squat during acceleration (Campbell, 1981; Sondkar & Jammulamadaka, 2021). Another type is the load-leveling suspension, which counteracts the tendency of the vehicle to be non-level, due to heavy cargo in specific locations (Elmadany, 1990). Yet another is designed to provide isolation from high-frequency shock (Islam & Ahmed, 2006). Other design considerations include the space occupied, with MacPherson struts being a typical compact arrangement (Hales, 1964; Kodati, Reddy & Bandyopadhyay, 2015; Reddy, Kodati, Chatra & Bandyopadhyay, 2016), the force distribution, air resistance—due to the airflow around any exposed suspension components—and of course the cost.

One advantage of the leaf spring is its ability to be attached directly to the chassis. This may contribute to its status as one of the oldest and most widely used springs in suspension systems. The leaf spring's characteristics can be changed to match the requirements of the design by changing the width, thickness, and length of the leaves of the springs. In addition, the lead spring provides a damping effect, due to friction in the mechanism. The torsion bar design, in which the spring is simply a round bar designed to twist as a force is applied to the suspension, is used in some vehicles, depending on the space constraints available. Typically, the torsion bars are approximately four feet long, with the tension in the bar controlled by a threaded screw adjustment. Coil springs occupy a relatively small space and, therefore, can be used in a variety of suspension designs, including the MacPherson strut, a solid axle with a trailing arm, independently sprung rear axle, or any short-long-arm (SLA) suspension system using a spring or coil-over shock absorber configuration. The coil spring's characteristics are determined by the wire gauge, spring length, overall diameter of the spring, and the number of coils. Coil springs can also have a variable rate, with the load-bearing capability increasing as it is compressed; such springs are often used in chassis configurations that occasionally carry heavy loads.

In this current study, the authors decided to focus on coil spring suspensions and conducted a review of the existing literature and research on this specific aspect of suspensions; they found that much of it focuses on manufacturing and design. In a study by Serbino and Tschiptschin (2011), the authors focused on fatigue, in particular the benefits of

austempering of springs and the benefits of this process over quenched springs, in order to better handle cyclic loads. This specific paper focused on springs in automotive valves, but the loading conditions and mechanisms were similar for suspension springs, albeit with different magnitudes of the loads. Virnich and Muhr (1985) focused on fatigue, but centered their work specifically on suspensions. In that study, the authors compared different alloys and found a new Si-Cr-V steel alloy to be effective. Vaillant and Ferlicca (1985) studied the impact of a variable diameter or tapered spring, and found it to be beneficial for packing reasons, though it did present a cost increase.

Sakakibara, Kusakari, Nakano, Yasuda, Sugimoto and Watanabe (1993) also addressed the questions of materials and manufacturing, with a FE-C-Si-Mn-Ni-Cr-Mo-V alloy that was able to provide weight reduction due to its improved mechanical properties. Miyamura, Kunou, Saitoh, Matsumoto, Yamamoto, Tsurui, and Homma (1993) performed a study on ovate wire helical springs, specifically looking at two types of wire cross sections: one elliptical and one a Fuch's egg-shaped cross section. This work indicated that the elliptical spring was superior. And, finally, Pawar, Patil and Zope (2016) conducted the design and analysis of a coil spring specifically for the front suspension of a three-wheel vehicle. This work involved theoretical analysis and software simulations, including finite element analysis, in order to ensure that the design requirements were met.

Methods

In order to improve an automotive suspension, the authors of this current study decided that the specific design to be considered would be one with a coil spring, as they are a common configuration and that the specific component to be studied would be the spring itself, as it has a large impact on the overall suspension. The spring must be designed such that it has a spring constant that leads to good performance, but it also must be able to withstand the stress imposed by the loads on it. Figure 1 shows a schematic of these loads.

Equations 1 and 2 give the maximum stress and maximum displacement for the spring, respectively (Edwards and McKee, 1991):

$$\sigma_{\max} = \frac{8K_s FD}{\pi d^3} \tag{1}$$

$$\delta_{\max} = \frac{8K_s NFD^3}{Gd^4} \tag{2}$$

where, D is the diameter of the coil, *d* is the diameter of the wire, L is the length of the spring, F is the load, K_s is the shear stress correction factor, G is the shear modulus of elasticity of the material, and N is the number of active coils.

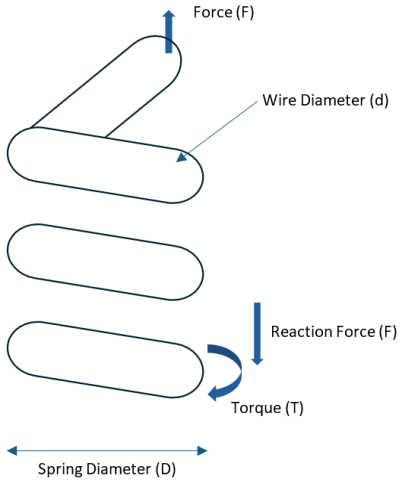


Figure 1. Schematic of loading on a coil spring.

Under typical loading conditions, stress is greatest at the inner portion of the spring. In considering the loading conditions, the authors assumed that the weight distribution on each wheel was nominally one fourth of the vehicle weight at rest, and that through weight shifts in maneuvers it could increase to as much as double that, or half of the vehicle weight. This was judged to provide a substantial margin of safety, as a vehicle maneuver that would put half the weight on a single wheel would be quite aggressive. Based on the range of weights seen for a variety of different types of vehicles, three values were chosen for the final loading, at 2015 N, 4320 N, and 5150 N (Dave, 2018). The analysis was conducted using the Siemens NX software package. The procedure for doing so was as follows:

Step 1: Generation of the CAD model for the spring, including full constraints.

In this step, the 3D model was created and structured in such a way that it could be easily changed. This involved the relationship between key dimensions of the spring, ensuring that minimal independent changes would be needed to create a new spring, and that impossible configurations (e.g., ones where the wire diameter was greater than the spring diameter) would not occur. The loading and constraints were placed on the top and bottom surfaces. Figure 2 shows a CAD model of the spring (Dave, 2018).

Step 2: Analysis Setup

At the end of the first step, the part would be in the modeling section of the program. To set up the analysis, the pre-post section was selected and used. A new finite element model (FEM) and simulation was started, and appropriate settings were selected. These included unchecking the "idealized part" option, selecting the solution type as SOL 101 Structural Analysis, and the solver type as NX Nastran. This then generated three files: a *.prt part file, a *.fem finite element method file, and a *.sim simulation file.

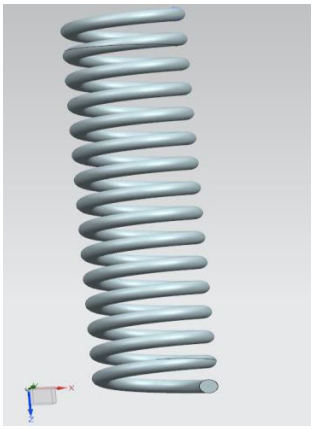


Figure 2. CAD model of the spring (Dave, 2018).

Step 3: Setting up the Finite Element Modeling File

The next step involved setting up the FEM file. A material was selected for the spring from a list of options, and then a mesh was chosen. A tetrahedral mesh was applied to the part. While the software is capable of estimating mesh size, in order to ensure that the mesh was not too large, the recommended size was reduced by a factor of two. Figure 3 shows the meshed spring (Dave, 2018).

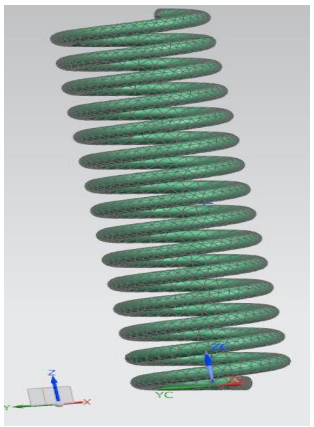


Figure 3. Meshed spring model (Dave, 2018).

Step 4: Setting up the Simulation File

Setting up the simulation file involved providing loads and constraints. The load category selected was "force," and the load value in Newtons was specified, in accordance with the three load values listed above. Figure 4 shows how this load was applied to one of the divided faces (Dave, 2018), which was the face that was assumed to be directly connected to the vehicle chassis or upper control arm.

Constraints were also applied as part of the simulation file setup. A fixed constraint was applied to the other side of the divided face of the spring, which was assumed to be connected to the axle or the lower control arm. Only vertical linear motion of the top of the spring was allowed by the constraint that was applied to the top of the spring, which was a user-defined constraint. Figure 5 shows the fully constrained file (Dave, 2018).

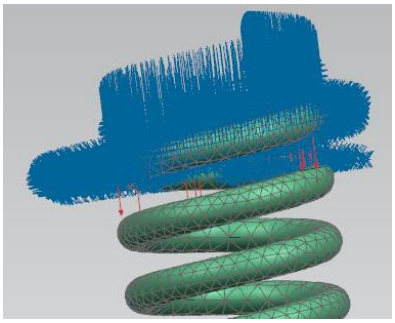


Figure 4. Forces (red arrows) applied to the top of the spring (Dave, 2018).

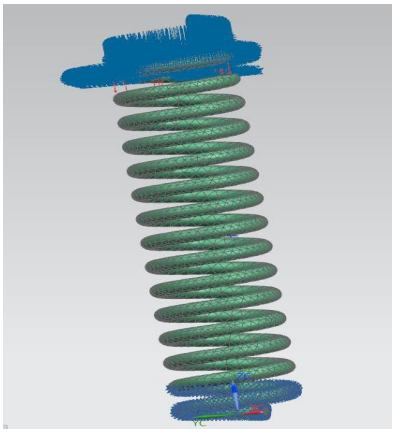


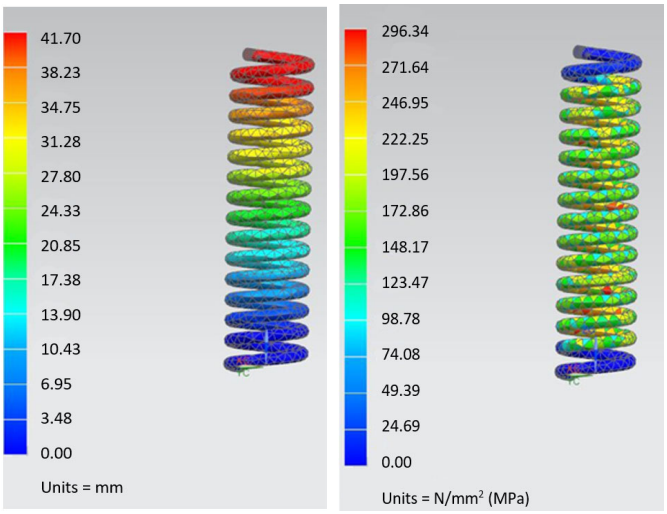
Figure 5. Fully constrained spring (Dave, 2018).

Step 5: Analysis

In the final step, the analysis was carried out for each of the 36 cases considered in this study. This consisted of simply running the analysis that had been set up in these steps, which yielded results for both the stress and displacement. Figure 6 gives typical results (Dave, 2018).

Design Conditions

In the initial analysis plans, the authors intended to look at tapered springs as well as those with non-symmetric cross sections. However, as discussed in the Results section, these possibilities were eliminated, based on preliminary analyses that involved different possibilities for the coil diameter, wire cross section (shape and size), and applied load; the length of the spring was held constant at 450 mm. A coding scheme was used to designate the springs, where the first two characters indicated coil diameter, the next two indicated the wire cross section, and the final two characters indicated the loading condition. Table 1 provides the values corresponding to each code.



a) Deformation b) Stress

Figure 6. Typical results of analysis (Dave, 2018).

Initially, it was intended that the symmetric ovate should utilize both d1 and d2 diameters, but preliminary analysis resulted in the symmetric ovate being limited to the d2 diameter, as noted in the Results section, hence resulting in three options for coil diameter, three for applied load, and four for wire cross section, or 36 total combinations. These combinations were coded as described previously, such that a design condition designated as D2d1L3 would indicate that the coil diameter was 100 mm, the wire cross section was circular with a diameter of 16 mm, and the applied load was 5150 N.

Table 1. Conditions for analysis.

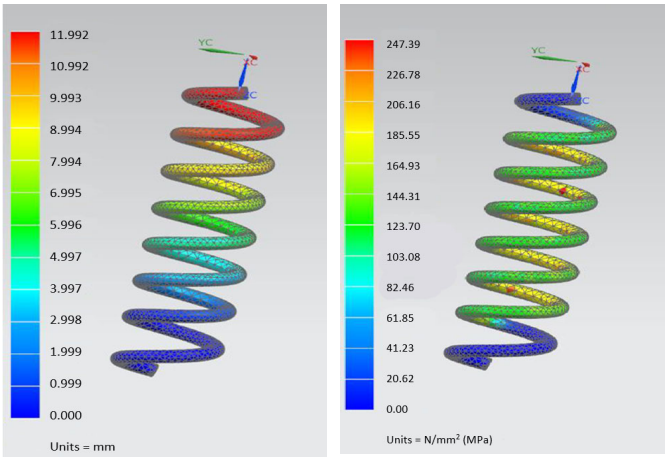
Coil Diameter		D1	80 mm
		D2	100 mm
		D3	120 mm
Wire Cross Section	Circular	d1	16 mm
		d2	18 mm
	Symmetric Ovate	S1	1.5d
		S2	2d
Applied Load		L1	3015 N
		L2	4320 N
		L3	5150 N

Subsequently, the best choices of springs for two conditions, one of low displacement and one of high displacement, were subjected to optimization, with the length and pitch of the spring chosen as design variables. The objective for the optimization was to minimize weight, with constraints on the stress and displacement. Finally, the designs that were optimized were analyzed to determine whether they would withstand fatigue loads. This analysis was carried out for three different alloys: Fe-C-Si-Mn-Ni-Cr-Mo-V, Si-Cr-V, and Cr-V. Material properties for these alloys can be found in the study by Sakakibara et al. (1993). In that analysis, the authors assumed that the life of the spring should be at least 2 x 10^5 cycles in order to be considered acceptable. This analysis was also carried out in NX, using the software's tools for fatigue analysis in the "durability wizard" provided in the software. In setting up the analysis, the yield strength was selected as the stress criteria, with the stress type selected as Von Mises. The solution type was selected as SOL103, and the remainder of the finite element setup was the same process as for the stress analysis (i.e., Step 3). Instead of an applied load, however, an enforced motion constraint was used to cycle the spring.

Analysis and Results

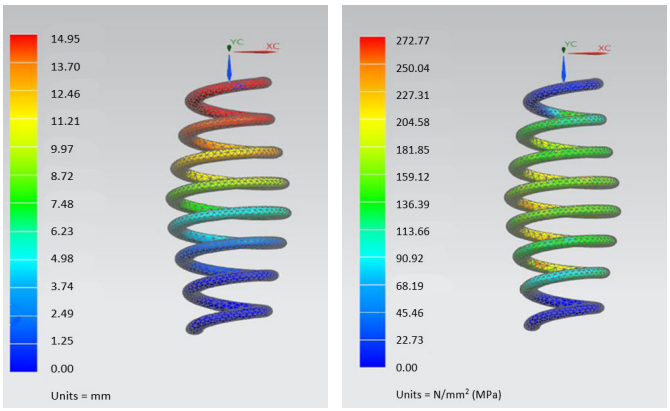
Initial analysis was conducted on the full range of possible designs that would be present if the conditions included a tapered spring, non-symmetric ovate cross section, and a symmetric ovate with both d1 and d2 used for their cross sections for the value of *d*, in addition to the conditions in Table 1. Test runs were conducted, with the goal of determining whether some possible designs could be eliminated outright from consideration; it was found that this was in fact the case. As stated in the literature, an ovate cross section reduces the stress value dramatically. It was hypothesized that an ovate section, based on the value of d1, could be eliminated, since, if a spring using d1 for a circular cross section was an optimum design, then an ovate cross section based on that value would be under-designed. It was further reasoned that a spring using d2 would provide better results than one using d1, independent of what other values were selected, and this was tested for multiple cases for verification. The non-symmetric section has a comparatively blunt side on one end, compared to the other. It was predicted that the blunt end would provide a reduction of stress concentration. Therefore, before attempting final analysis, a comparison was made between non-symmetric and symmetric designs.

After performing analyses on several samples, it was seen that the stress values in non-symmetric springs were higher than those for symmetric springs. Therefore, it was decided that the use of the non-symmetric design would not satisfy the aim of the project; hence, this design condition was dropped from consideration. In fact, analysis showed that the non-symmetric ovate design resulted in stress values that were almost double in some cases. A number of tapered spring designs were included in the initial test cases; however, it was noted that those designs invariably had high stresses compared to corresponding designs with a constant coil diameter. This particular spring design was meant to provide a reduction in length, but it was judged that this did not provide a significant advantage in the context of this study. Figures 7 and 8 show comparisons of stress results for the constant diameter and tapered springs, respectively (Dave, 2018). In this case, there was little difference, but also little advantage, and thus the tapered springs were not included in the systematic analysis.



a) Displacement b) Stress

Figure 7. Constant diameter spring (Dave, 2018).

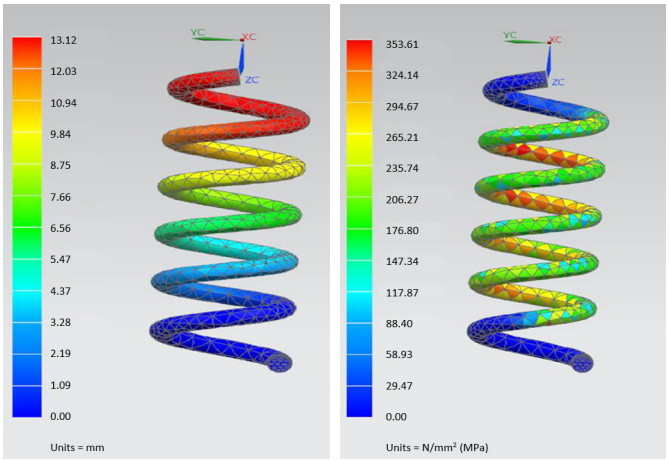


a) Displacement b) Stress

Figure 8. Results for tapered spring (Dave, 2018).

Analysis of the Set of 36 Cases

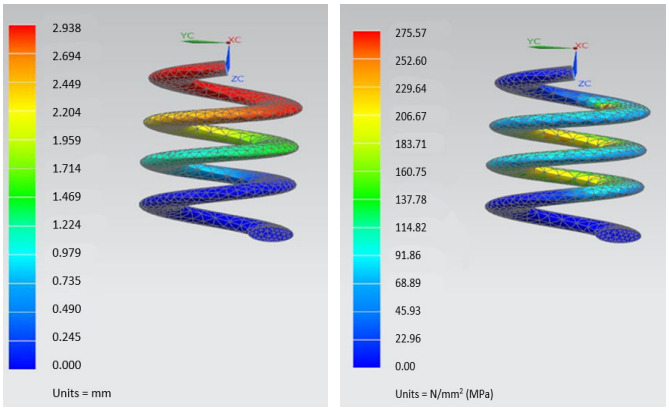
When the 36 cases (described in the Methods section) were analyzed, each was categorized based on their safety factor. While a safety factor had already been built into the problem via the assumption of half the vehicle weight being on one tire, an additional safety factor was considered, due to simplifications and uncertainties in the problem. If the safety factor was less than 2, then the design was considered as one that may fail. Those with safety factors between 2 and 3 were considered to be marginal, and those with a safety factor of 3 were considered to be optimal. Two types of designs were considered for further consideration, based on either high displacement or low displacement. Table 2 gives the full set of results (Dave, 2018); note that red denotes values at the stress limit, white rows represent failing designs, and tan are the marginal designs. The designs shown in green are the springs with a high displacement, while those shown in blue have a low displacement. The spring D1d2L1 was chosen as the starting point for the high -displacement version of the suspension. After carrying out the optimization procedure described in the Methods section and then checking the stress and displacement to ensure that constraints were not violated, it was found that the pitch of the spring had changed from 30 mm to 60 mm, and the length changed from 450 mm to 346 mm. Figure 8 shows the displacement and stress for this design (Dave, 2018).



a) Displacement b) Stress

Figure 9. Results for spring D1d2L1 with a pitch of 60 mm and a length of 346 mm (Dave, 2018).

For the low-displacement version of the design, the spring D2S2L2 was used as a starting point. When the optimization was carried out, the pitch of the spring was set at 60 mm and the length became 225 mm. Figure 10 shows the displacement and stress for this design (Dave, 2018).



a) Displacement b) Stress

Figure 10. Results for spring D2S2L2 with a pitch of 60 mm and a length of 225 mm (Dave, 2018).

In the final step, where a fatigue analysis was carried out, the authors found that the best choice of material for fatigue life was the Cr-V alloy, followed by the Si-Cr-V alloy, with the last choice being the Fe-C-Si-Mn-Ni-Cr-Mo-V alloy.

Table 2. Summary of design analysis results (Dave, 2018).

	Displacement (mm)	Stress (Mpa)	Bumping stress	Bumping Disp.	Total stress	Total Disp
D1d1L1	41.7	296.34	29.85	3.76	326.19	45.46
D1d1L2	59.75	424.6				
D1d1L3	71.23	506.18				
D1d2L1	26.44	230.91	25.62	2.388	256.53	28.828
D1d2L2	37	330.86	25.25	2.384	356.11	39.384
D1d2L3	45.16	394.43	23.39	2.356	417.82	47.516
D1s1L1	11.89	139.53	17.24	2.863	156.77	14.753
D1s1L2	17.05	199.92	16.96	2.861	216.88	19.911
D1s1L3	20.33	238.33	17.09	2.746	255.42	23.076
D1s2L1	7.16	103.94	10.413	1.633	114.353	8.793
D1s2L2	10.26	148.93	12.26	1.7	161.19	11.96
D1s2L3	12.23	177.55	10.278	1.701	187.828	13.931
D2d1L1	56.85	366.05	37.25	4.87	403.3	61.72
D2d1L2	81.46	524.49				
D2d1L3	97.11	625.26				
D2d2L1	35.27	254.08	27.56	3.176	281.64	38.446
D2d2L2	50.54	364.05	27.69	3.62	391.74	54.16
D2d2L3	60.25	434				
D2s1L1	16.48	152.82	17.82	2.765	170.64	19.245
D2s1L2	23.61	218.96	17.8	2.746	236.76	26.356
D2s1L3	28.14	261.03	17.88	2.653	278.91	30.793
D2s2L1	10.07	110.98	10.446	1.7	121.426	11.77
D2s2L2	14.43	159.02	13.2	1.716	172.22	16.146
D2s2L3	17.2	189.57	12.75	1.698	202.32	18.898
D3d1L1	97.76	394.99	25.2	4.97	420.19	102.73
D3d1L2	140.07	565.96				
D3d1L3	166.99	674.69				
D3d2L1	60.94	300.58	28.84	5.25	329.42	66.19
D3d2L2	87.31	430.68				
D3d2L3	104.09	513.42				
D3s1L1	16.48	152.82	17.74	2.653	170.56	19.133
D3s1L2	23.61	218.96	17.96	2.652	236.92	26.262
D3s1L3	28.14	261.03	17.12	2.633	278.15	30.773
D3s2L1	17.97	132.37	10.559	1.699	142.929	19.669
D3s2L2	25.74	189.67	13.3	1.715	202.97	27.455
D3s2L3	30.69	226.11	12.7	1.629	238.81	32.319

Conclusions

In this study, the authors subjected the springs in an automotive suspension to analysis and optimization. The procedure used here involved first analyzing multiple cases, then selecting the best ones for further optimization and a final analysis for fatigue life. It was found that changes in the springs could result in improved suspensions, with the definition of improvement depending on whether a high or low displacement was chosen for the suspension. As vehicle types and the desired suspension characteristics are complex, this leads to many options for future work.

Future work could include both further focus on the springs themselves as well as a more holistic look at the suspensions in which they are used. An optimization procedure that includes the coil diameter, wire diameter, length, and pitch might produce better results than those seen here, albeit with the added complexity of a larger optimization problem. Furthermore, an optimization that includes the design characteristics of the springs as well as other components of the suspension could result in finding synergies between different elements of the suspension design, which may provide even more improvements. In addition, in the design and analysis of a more extensive suspension, an active suspension with a controller could be implemented, with a co-design approach to the design and control, like that set forth in the study by Peters et al. (2013). This future work could also be tested through production of the designs and physical testing, in order to validate the theoretical results found through this work and its extensions. In addition, future work should consider a detailed analysis of vehicle dynamics, including handling and comfort. Such an analysis should include a full model of the suspension and require multiple test cases to adequately cover the range of operating conditions.

References

Bartolozzi, R., & Frendo, F. (2011). Stiffness and strength aspects in the design of automotive coil springs for McPherson front suspensions: A case study. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, 225*(10), 1377-1391.

Baviskar, A. C., Bhamre, V. G., & Sarode, S. S. (2013). Design and analysis of a leaf spring for automobile suspension system: a review. *International Journal of Emerging Technology and Advanced Engineering, 3*(6), 407-410.

Campbell, C. (1981). *Suspension Geometry. In Automobile Suspensions*. Boston, MA: Springer US.

Dave, V. (2018). *Experimentation for Design Improvements for Coil spring in the Independent suspension*. Master's Thesis. Kettering University, Flint, MI.

De Melo, F. J., Pereira, A. B., & Morais, A. B. (2018). The simulation of an automotive air spring suspension using a pseudo-dynamic procedure. *Applied Sciences, 8*(7), 1049.

Edwards, K. S. Jr., & McKee, R. B. (1991). *Fundamentals of Mechanical Component Design*. New York, NY: McGraw-Hill, Inc.

Elmadany, M. M. (1990). Ride performance potential of active fast load leveling systems. *Vehicle System Dynamics, 19*(1), 19-47.

Hales, F. D. (1964). A theoretical analysis of the lateral properties of suspension systems. *Proceedings of the Institution of Mechanical Engineers: Automobile Division, 179*(1), 73-97.

Islam, A. S., & Ahmed, A. K. W. (2006). A comparative study of advanced suspension dampers for vibration and shock isolation performance of road vehicle. *SAE Transactions*, 302-311.

Joo, F. R. (1991*). Dynamic analysis of a hydropneumatic suspension system* (Unpublished doctoral dissertation, Concordia University).

Karuppiah, P. S., Ganesan, P., Kasavan, P. S., & Sambasivam, S. (2023). Experimental and simulation investigation of vertical response of stepped torsion bar spring for light motor vehicle suspension system. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, 237* (7), 1479-1488.

Kodati, M., Reddy, K. V., & Bandyopadhyay, S. (2015, June). Kinematic analysis of MacPherson strut suspension system. In TrC-IFToMM *Symposium on Theory of Machines and Mechanisms*, Izmir, Turkey.

Lavanya, N., Rao, P. S., & Reddy, M. P. (2014). Design and analysis of a suspension coil spring for automotive vehicle. *International Journal of Engineering Research and Applications, 4*(9), 151-157.

Lijun, Z., Zengliang, Y., & Zhuoping, Y. (2010). A novel empirical model of rubber bushing in automotive suspension system. Proceedings of the 10th international conference on motion and vibration control. https://doi.org/10.1299/jsmemovic.2010._3A12-1

Mahanthi, D. L., & Murali, C. V. S. (2017). Design and analysis of composite Leaf Spring for light Weight Vehicle. *International Journal of Advanced Engineering Research and Science, 4*(3), 237088.

Miyamura, N., Kunou, T., Saitoh, K., Matsumoto, Y., Yamamoto, H., Tsurui, Y., & Homma, S. (1993). *Design and Testing of Ovate Wire Helical Springs* (No. 932891). SAE Technical Paper.

Pawar, M. H. B., Patil, A. R., & Zope, S. B. (2016). Design and analysis of a front suspension coil spring for three wheeler vehicle. *International Journal of Innovations in Engineering Research and Technology, 3*(2), 1-6.

Peters, D. L., Papalambros, P. Y., & Ulsoy, A. G. (2013). Sequential co-design of an artifact and its controller via control proxy functions. *Mechatronics 23*, 409-418.

Reddy, K. V., Kodati, M., Chatra, K., & Bandyopadhyay, S. (2016). A comprehensive kinematic analysis of the double wishbone and MacPherson strut suspension

systems. *Mechanism and Machine Theory, 105*, 441-470.
Riduan, A. F. M., Tamaldin, N., Sudrajat, A., & Ahmad, F. (2018). *Review on active suspension system*. In SHS Web of Conferences (Vol. 49, p. 02008). EDP Sciences.
Sakakibara, T., Kusakari, W., Nakano, O., Yasuda, S., Sugimoto, A., & Watanabe, M. (1993). *Development of a new light-weight suspension coil spring* (No. 930263). SAE Technical Paper.
Serbino, E. M., & Tschiptschin, A. P. (2011). *Review on the fatigue mechanisms in automotive valve springs* (No. 2011-36-0003). SAE Technical Paper.
Sondkar, P., & Jammulamadaka, A. (2021). *A study of influence of suspension on driveline torque and evaluation of vehicle anti-squat/dive characteristics using a planar vehicle dynamics model* (No. 2021-01-0693). SAE Technical Paper.
Tavares, R., Molina, J. V., Al Sakka, M., Dhaens, M., & Ruderman, M. (2019). Modeling of an active torsion bar automotive suspension for ride comfort and energy analysis in standard road profiles. *IFAC-PapersOnLine, 52*(15), 181-186.
Tseng, H. E., & Hrovat, D. (2015). State of the art survey: active and semi-active suspension control. *Vehicle system dynamics, 53*(7), 1034-1062.
Vaillant, C., & Ferlicca, R. (1985). A new design and manufacturing process for suspension coil springs. *SAE transactions*, 305-314.
Virnich, K. H., & Muhr, K. H. (1985). Properties of cold coiled automotive suspension springs from high strength Si-Cr-V wire. *SAE transactions*, 287-297.

Biographies

DIANE L. PETERS is an Associate Professor of Mechanical Engineering at Kettering University. She earned her BS in mechanical engineering from the University of Notre Dame in 1993, MS in mechanical engineering in 2000 from the University of Illinois—Chicago, and PhD in mechanical engineering in 2010 from the University of Michigan. Dr. Peters' research interests include control co-design, control systems, and autonomous vehicles. Dr. Peters may be reached at dpeters@kettering.edu

YAOMIN DONG is a Professor of Mechanical Engineering at Kettering University. He received his PhD in Mechanical Engineering at the University of Kentucky in 1998. Dr. Dong has extensive R&D experience in the automotive industry and holds multiple patents. Dr. Dong's areas of expertise include metal-forming processes, design with composite materials, computer graphics, computer-aided engineering, and finite element analysis. Dr. Dong may be reached at ydong@kettering.edu

VIRAJ DAVE earned his MS in automotive systems from Kettering University in 2018. Mr. Dave may be reached at dave0651@kettering.edu

Deep Neural Networks and Universal Approximators II

Ying Liu, Savannah State University; Majid Bagheri, Savannah State University;
Antonio Velazquez, Savannah State University; Asad Yousuf, Savannah State University

Abstract

There are many studies of approximations using deep neural networks. In this paper, the authors provide yet another proof that deep neural networks are universal approximators. In their earlier work, the authors showed that an arbitrary binary target function can be effectively rewritten in terms of a set of strings, or a set of subsets, and that a single hidden neuron can identify and only identify a single string or a single subset. Therefore, an arbitrary binary target function can be effectively rewritten in the form of a neural network with one hidden layer. In this study, the authors imposed locality on the deep neural network, and will show here that an arbitrary binary target function can be effectively rewritten in the form of a locally connected deep neural network that can have many hidden layers. Although it will increase the neural network size, as a neural network is localized, it will generally increase the speed of training for large networks.

Key words: AI, Universal Approximator, Completeness, Deep Neural Network, Machine Learning, Supervised Learning, Unsupervised Learning, Locally Connected.

Introduction

Neural networks provide good solutions to many supervised learning problems. Neural networks have a long history, but there have been two main developments in recent years, deep learning and transforming (Amari, Kurata & Nagaoka, 1992; Byrne, 1992; Kubat, 2015). In 2006, authors in several other studies introduced the idea of "deep" neural networks (Hinton, Osindero & Teh, 2006; LeCun, Bengio & Hinton, 2015; Bengio, 2009; Coursera, 2017; Bengio, Courville & Vincent, 2013; Schmidhuber, 2015; Ciresan, Meier & Schmidhuber, 2012). Examples of software include TensorFlow (TensorFlow, 2017), Torch (Torch 2017), and Theano (Theano, 2017). Layers in deep neural networks (DNNs) serve as the building blocks of the architecture, enabling the model to learn from the data. Each layer has a specific function in transforming the input into an output, progressively extracting higher-level features. For example, early layers might detect edges or simple patterns (e.g., in images), while middle layers may capture more complex patterns (e.g., shapes or textures), and deeper layers identify task-specific, high-level features (e.g., faces or objects). The transformer model, introduced by Vaswani et al. (2017), represents a significant advancement in deep learning architectures, particularly in natural language processing. Each layer in the DNN is replaced by a transformer (Vaswani et al., 2017).

The transformer leverages a fully self-attentive mechanism to model complex dependencies between elements of a sequence. This architecture enables the transformer to be trained more efficiently and with greater parallelism, leading to faster training times and improved scalability. As a result, the transformer has become the backbone of numerous state-of-the-art models, including Chat GPT (OpenAI, 2023) and Claude (Anthropic, 2023), profoundly influencing the development of modern deep learning systems. Studies of the neural network as universal approximators have a long history. Hornik, Stinchcombe, and White (1989) established models showing that multi-layer feed-forward networks with hidden layers using arbitrary squashing functions are capable of approximating any measurable function from one finite dimensional space to another to any desired degree of accuracy, provided that a sufficient number of hidden units are available. In this sense, multi-layer feed-forward networks are a class of universal approximators.

Hinton, Osindero, and Teh (2006) introduced the idea that deep belief networks (DBN) are generative neural network models with many layers of hidden explanatory factors, along with a greedy layer-wise unsupervised learning algorithm. The building block of a DBN is a probabilistic model called a restricted Boltzmann machine (RBM), which is used to represent one layer of the model. Restricted Boltzmann machines are interesting, because they have been successfully used as building blocks for training deeper models. Le Roux and Bengio (2008) proved that adding hidden units yields a strictly improved modeling power, and that RBMs are universal approximators of discrete distributions.

Liu and Wang (Liu, 1993; Liu, 1995; Liu, 1997; Liu, 2002; Liu & Wang, 2018; Liu, 2018a/b) proved that DNNs implement an expansion and the expansion is complete; a complete expansion can be used to expand any target functions. Cheng, Li, and Lu (2019) introduced a type of convolutional neural network (CNN) that can implement the Fourier and local Fourier transformations for approximation in a large class of problems. Cybenko (1989) showed that a finite sum of any continuous sigmoid function can be used to approximate any univariate function using functional analysis. Liu and Yousuf (2020) showed that DNNs are effective universal approximators. An arbitrary binary target function can be effectively rewritten in the form of a DNN; thus, proving that DNNs can implement any target mappings. An example of a locally connected network is the convolutional neural network (CNN) (LeCun, Bottou, Bengio & Haffner, 1998; Krizhevsky, Sutskever & Hinton, 2012), which is a specialized type of deep learning model particularly well-suited for processing images.

It utilizes convolutional layers that apply filters across the input data to capture spatial hierarchies of patterns. This architecture allows the network to automatically learn and detect features such as edges, textures, and objects within images, making them highly effective for tasks such as image classification, object detection, and segmentation. Several approaches can be applied for reducing computation times of neural networks, including model optimization, hardware utilization, and algorithmic refinement. 1) Model optimization techniques, such as pruning, quantization, and knowledge distillation, reduce model size while maintaining performance (Han, Pool, Tran & Dally, 2015). Weight sharing and sparse representations are also effective in minimizing redundancy in parameters. 2) Efficient architectures, for example MobileNet (Howard et al., 2017) and EfficientNet (Tan & Le, 2019) were explicitly designed to reduce computational overhead through depth-wise separable convolutions and scaling strategies. 3) Hardware-specific optimizations are accelerators—such as GPUs, TPUs, and custom ASICs—that exploit parallelism and optimized memory access patterns to enhance speed (Jouppi et al., 2017). 4) Training techniques, such as mixed precision training (Micikevicius et al., 2018), reduce floating-point precision for faster computation, while learning rate schedulers and gradient accumulation ensure efficient convergence.

In this paper, the authors will show that an arbitrary binary target function can be effectively rewritten in the form of a locally connected DNN. The result opens a discussion for exploring an approach of locally connected neural networks as an alternative to globally connected models. Additionally, the authors will build on their earlier work (Liu & Yousuf, 2020); in particular, locality, will be imposed on the neural network. As a comparison, the earlier work had a single globally connected network with one hidden layer, while the work presented here represents many hidden layers with locally connected neural networks. The author will show that an arbitrary binary target function can be effectively rewritten in the form of a locally connected DNN. To prove this result: 1) the earlier work by the authors will be briefly reviewed foundational to this current study; 2) the result is proof for a special case—a binary locally connected network; and, 3) the result will be proven by removing the binary condition. For a given target function, there are many effective ways to construct a locally connected DNN.

The results, then, open a discussion for exploring an approach of locally connected neural networks as an alternative to globally connected models. The von Neumann bottleneck refers to the limitation in computing systems that stems from the separation of the central processing unit (CPU) and memory in the von Neumann architecture. Increasingly, both computation times and electric powers are spent on moving data from one place to another. For example, electric power consumption has been increasing rapidly for the transformer models. (Wall Street Journal, 2024). One of the main reasons that transformer models use far more power than biological neurons is that the biological systems are locally connected networks. In this paper, the authors show that, as the models transit from globally connected networks to locally connected networks, the computing power will not be decreased, but the amount of data transfer can be reduced.

Review: Effectively Rewriting a Mapping with One Hidden Layer

An arbitrary binary target function can be effectively rewritten in terms of a set of strings, or a set of subsets. A single string or a single subset can be identified by a single hidden neuron, and this neuron will only identify the string or the subset; therefore, an arbitrary binary target function can be effectively rewritten in the form of a neural network with one hidden layer (Liu, Yousuf, 2020). A binary-neuron input instance is 00 … 0, or, 0 … 01, … (Amari et al., 1992; Byrne, 1992; Kubat, 2015) and an instance space (Kubat, 2015) is a set of all instances given by Equation 1:

$$X = \{0\ldots00, 0\ldots01, 0\ldots10, 0\ldots11, \ldots\} \tag{1}$$

Given an arbitrary binary target function, it can be effectively rewritten in terms of a set of strings, or a set of subsets given by Equations 2 and 3:

$$h = \{s_0, s_1, s_2, \ldots\} \tag{2}$$

$$s_i \subseteq \{0, 1, 2, \ldots d-1\} \tag{3}$$

Example. Given a function in Table 1, the mapping can be rewritten using Equations 4 and 5:

Table 1. A sample binary function with three inputs.

x_0	x_1	x_2	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

$$y = \{001, 011, 100\} \tag{4}$$

$$y = \{\{2\}, \{1, 2\}, \{0\}\} \tag{5}$$

where, y is overloaded with a table, a mapping, a set of strings, and a set of subsets, and s_i in Equation 3 is overloaded with a string and a subset.

Without loss of generality, it can be assumed that there is only one output variable for now. For the case of multiple output variables, it can be treated as multiple mappings. The neural network used in this review section will have one input layer, one output layer, and one hidden layer. The neuron values are given by Equation 6 (Amari, Kurata & Nagaoka, 1992; Byrne, 1992; Kubat, 2015):

$$y_i = f\left(\sum w_{ij} x_j + b_i\right), i = 1, 2, 3, \ldots \tag{6}$$

where, f is a sigmoid function given define by Equation 7:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

To compute the connection weights, a constant L is introduced; without a loss of generality, set L = 10. For an arbitrary target function, it can be rewritten in the form of Equations 2 and 3. The rules for construction of a DNN are:

1. The DNN will have one input layer, one output layer, and one hidden layer. The input layer has d neurons.
2. Each neuron in the hidden layer identifies one string in a target function, h = {s_0, s_1, …, }, so the number of neurons in the hidden layer is |h|, which is the number of strings or the number of subsets.
3. The output layer has one neuron; the neuron value is 1, if any one of the hidden layer neurons is 1.
4. Assume that s is a subset in a mapping, h; and assume a hidden neuron will identify s; the subset, then, is given by Equations 8 and 9:

$$s = \{j_0, j_1, j_2, \ldots\} \tag{8}$$

$$s \subseteq \{0, 1, 2, \ldots d-1\} \tag{9}$$

The hidden neuron has weights and biases as follows:

set weight = L, for input neurons {j_0, j_1, j_2, …}
set weight = – L, for all other input neurons
set bias = - (|s| - 1) · L

It has been proven that this simple ANN will implement a target function (Liu & Yousuf, 2020). To summarize:

1. An arbitrary binary target function can be effectively rewritten in terms of a set of strings, or a set of subsets, given by Equations 2 and 3:
2. A single hidden neuron can identify and only identify a single string or a single subset. The weights and biases are directly determined from a given target function by the rules in this section.
3. An arbitrary binary target function can be effectively rewritten in the form of a neural network with one hidden layer.

Binary Locality or Bilocality

In a locally connected neural net, let the maximum number of connections of a hidden neuron be N. To simplify the discussion, let the locality be extreme: N = 2; from Figure 1, this is called binary locality or simply bilocality. Once N is restricted, the size of connection matrices is restricted, at the cost of increasing the number of matrices. This reduction of one large matrix into many smaller matrices has its implications in computation efficiency, especially when the matrix is very large.

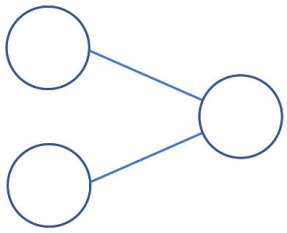


Figure 1. A hidden neuron has two input neurons.

The following naming convention will be adopted:

- The hidden layer closest to the input is the first hidden layer.
- The hidden layer closest to the output is the last hidden layer.

Assumption 1:
The DNN (deep neural network) is bilocally connected, where each hidden neuron can have only two or fewer connections.

Assumption 2:
The number of neurons in the input layer is a power of 2 (e.g., 2, 4, 8, 16, …).

These two assumptions will be removed later. Furthermore, Assumption 1 only applies to the hidden neurons; the connections of the single output neuron are determined by the number of strings in a target function in Equation 2, |h|. Without loss of generality, it can be assumed that there is only one output variable for now. For the case of multiple output variables, it can be treated as multiple mappings. In a binary locally connected network (bilocal network), it is only natural to group the connection weights of a neuron with the neuron rather than group them into connection matrices. A binary locally connected neural net is a set of neurons; a neuron has a neuron value, two connection weights, and a bias (called neuron-based computation): NN1 = {Neuron}, Neuron = {value, w0, w1, b}. This is in contrast to the view that a neural net is a set of neurons (where each neuron has a single value), a set of connection matrices, and a set of bias vectors (called matrix-based computation): NN2 = {Neurons, Matrix, Bias}. To compute the connection weights, a constant L will be introduced; without a loss of generality, set L = 10. A binary function is then rewritten in terms of a set of strings of 0's and 1's. A string in the set is directly imposed to the input neurons. For a bilocal network, two input neurons are grouped together and its two-bit pattern is passed to a hidden neuron in the first hidden layer.

Let a sample string be $x_0x_1x_2x_3$, where the pattern x_0x_1 can be identified by a hidden neuron, h_1, in the next layer, and the pattern x_2x_3 can be identified by a hidden neuron, h_2. The identification of x_0x_1 is propagated to the next layer via h_1, and the identification of x_2x_3 to h_2. To identify the entire pattern $x_0x_1x_2x_3$, h_1 and h_2 are further propagated to a hidden neuron in the next layer, say h_3, which only needs to identify the pattern "11" (i.e., both h_1 and h_2 have identified their required patterns). This is the basic idea of the newly proposed algorithm. The new rules for the network construction are:

1. The input layer has $d = 2^K$ neurons. The DNN has one input layer, one output layer, and O (log d) hidden layers.
2. Each neuron in the last hidden layer identifies one string in a target function, $h = \{s_0, s_1, \ldots, \}$, so the number of neurons in the last hidden layer is |h|, which is the number of strings in Equation 2 or the number of subsets.
3. The output layer has one neuron; the neuron value is 1, if any one of the last hidden layer neurons is 1.

Rule 1 states that there are *d* input neurons. The condition, $d = 2^K$, is for the sake of easy discussion and will be removed later. Rule 2, together with several other rules, describes the overall hidden neuron structures; each layer has a specific function in transforming the input into an output, progressively identifying bigger bit patterns for strings in a target function. In particular, Rule 2 specifies the last hidden layer, and its role is: a) the number of hidden neurons in the last hidden layer is the same as the number of strings in a given target function, and b) each hidden neuron in the last hidden layer will identify and only identify one string in the target function. Rule 3 describes the output layer. For the sake of this discussion, assume that there is only one output variable, per our earlier assumption, so there is only one output neuron. If an input string is one of the strings in a target function, one of the hidden neuron values in the last hidden layer is 1, which will cause the output neuron to be 1. If an input string is not in the target function, all of the hidden neurons in the last hidden layer will be 0, which will cause the output neuron to be 0.

Single String Identification

To identify a single string or a single subset, let the input layer have *d* neurons; let the first hidden layer have d/2 hidden neurons; let the second hidden layer have d/4 hidden neurons; and, let the last hidden layer have one hidden neuron. The input layer and all hidden layers together then form a binary tree, called a hidden tree. A hidden tree will identify one string in a target function later. In a complete binary tree, there exist relationships between the height, the number of edges, and the number of nodes in each layer from which a complete binary tree has:

- *d* input neurons
- log(d) hidden layers
- 2d – 2 weights
- d – 1 hidden neurons

By way of example, Figure 2 shows a hidden tree that has:

- d = 4 input neurons
- log(d) = 2 hidden layers
- 2d – 2 = 6 weights
- d – 1 = 3 hidden neurons

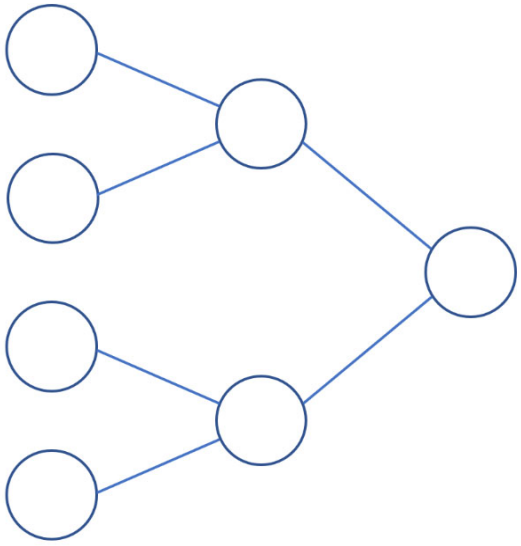


Figure 2. An example of a hidden tree with 4 input neurons.

In a complete binary tree, there exist relationships between the height, the number of edges, and the number of nodes in each layer. The four input neurons are drawn in column 1; thus, d = 4. Shown in Figure 2, the number of hidden layers is log (d) = 2: column 2 and column 3. Also shown in Figure 2 as edges is the number of weights: 2d – 2 = 6. The number of hidden neurons is d – 1 = 3: the 3 nodes in columns 2 and 3. This is the tradeoff between a globally connected network and a locally connected network. There are two costs of locality: 1) from using a single hidden neuron to identify a single string in a fully connected network to d – 1 neurons; 2) from using *d* weights to identify a single string in a fully connected network to 2d – 2 weights.

The first hidden layer identifies the input patterns. Each neuron in the first hidden layer identifies two input bits (bilocal). The number of neurons in the first hidden layer has d/2 neurons. After all of the 2-bit patterns are identified, the results propagate up, eventually to one single neuron in the last hidden layer. The role of the first hidden layer is to identify a single string or a single subset, and the roles of the rest of the hidden layers are to pass the results of the first hidden layer to a single root of a hidden tree in the last hidden layer.

Target Function Identification

Each neuron in the last hidden layer identifies one string in a target function, $h = \{s_0, s_1, \ldots, \}$, so the number of neurons in the last hidden layer is |h|, which is the number of strings or

the number of subsets. Figure 3 shows how each of the neurons in the last hidden layer grows a binary tree all the way to the input neurons.

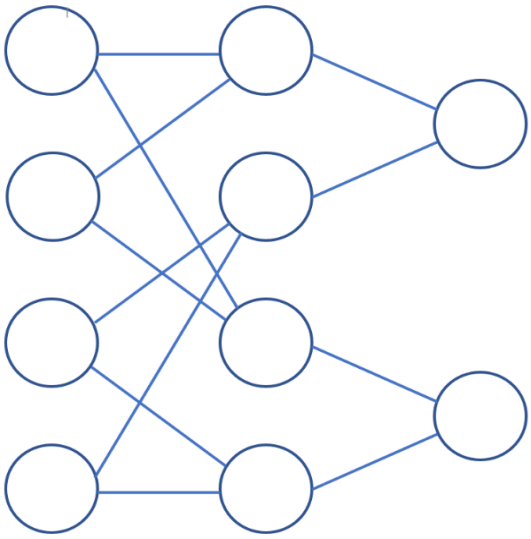


Figure 3. An example of two hidden trees for two strings.

In this example, the input layer has d = 4 input neurons and the target function has two strings to be recognized. There are two neurons in the last hidden layer; each is responsible for identifying one string. Each of the two neurons in the last hidden layer forms a binary tree. Within each tree, there are d – 1 hidden neurons in log(d) hidden layers and *d* input neurons. The rules for the hidden trees are:

4. Each of the neurons in the last hidden layer grows a binary tree all the way to the input layer neurons. There are log(d) hidden layers, where *d* is the number of input neurons. There are (d – 1) hidden neurons in each hidden binary tree.
5. The first hidden layer identifies the input patterns. Each neuron in the first hidden layer identifies two input bits (bilocal).

Let *s* be a single subset that is given in Equations 2 and 3, such that the rule for neurons in the first hidden layer is:

6. Assume that *s* is a subset in a mapping, *h*; further assume that a hidden neuron identifies *s*. In this case, the subset is given by Equations 8 and 9:

The hidden neuron has weights and biases as follows:

set weight = L, for input neurons $\{j_0, j_1, j_2, \ldots\}$
set weight = – L, for all other input neurons
set bias = $-(|s| - 1) \cdot L$

After all of the 2-bit patterns are identified in the first hidden layer, based on the rules above, the results will propagate up, eventually to one single neuron in the last hidden layer for one string/subset in Equations 2 and 3. The rule for neurons in the rest of the hidden layers is:

7. For the rest of the hidden layers (other than the first), all connection weights are L and all biases are $-(|s| - 1) \cdot L$, which is -L for bilocal hidden neurons.

This is an effective construction of a bilocal DNN from a given target function, which will be justified in the next section.

Effectively Rewriting a Mapping in Terms of a Bilocal Deep Neural Network

In the earlier review section, it was noted that a 2-bit pattern can be identified correctly by a hidden neuron. In this paper, the authors first identify a 2-bit pattern by one neuron in the first hidden layer, which has been proven to be correct. Second, the above step is repeated for all 2-bit patterns in the input layer. For d-input neurons, there are d/2 neurons in the first hidden layer. This step is already different from the authors' previous study in which they used one neuron instead of d/2 neurons. Third, the results of the first hidden layer simply propagate up. Let h_1 and h_2 be two neurons in the first hidden layer, the weights and the biases of a bilocal neuron, h_3, in the next hidden layer are simply (L, L), and -L, respectively, which identify the pattern "11" (i.e., both h_1 and h_2 have identified their required patterns). Each neuron in the second hidden layer identifies a 4-bit pattern. Fourth, each neuron in the third hidden layer identifies an 8-bit pattern, …, eventually, each neuron (root of a hidden tree) in the last hidden layer identifies one string or one subset. Finally, the output layer has one neuron; the neuron value is 1, if any one of the last hidden layer neurons is 1.

Since bilocal neurons are so simple, details can be worked out from the beginning in just a few lines. One bilocal hidden neuron is given in Equation 10:

$$m = f\left(a_0 x_0 + a_1 x_1 + b\right) \tag{10}$$

where, *f* is given in Equation 7.

Each hidden neuron in the first hidden layer has two weights and there are d/2 neurons. Since there are only four possible patterns to be identified, Table 2 lists the parameters in Equation 10 required for each case.

Table 2. Bilocal hidden neuron parameters for all 2-bit identifications.

Pattern to be identified	a_0	a_1	b
00	-L	-L	L
01	L	-L	0
10	-L	L	0
11	L	L	-L

Take, for example, one instance in detail. Assume that a neuron, *m*, can identify a pattern, "10"; from Table 2, Equation 10 is changed to Equation 11. All possible inputs and outputs for Equation 11 are listed in Table 3.

$$m = f\left(-Lx_0 + Lx_1\right) \tag{11}$$

Table 3. Inputs and outputs for Equation 11.

Input	$-Lx_0 + Lx_1$	m	int (m)
00	0	1/2	0
01	-L	0	0
10	L	1	1
11	0	1/2	0

Column 1 shows all possible inputs for 2-bit patterns. Column 2 is the intermediate step. Column 3 shows the neuron values. Column 4 takes the integer part of Column 3. In Table 3, int(m) is the integer function in C# language. The hidden neuron identifies the correct string, "10", by Equation 12:

$$m = f\left(-Lx_0 + Lx_1\right) = \frac{1}{1+e^{-L}} \approx 1 \tag{12}$$

If there is a single bit difference ("00", "11"), the hidden neuron has a value given by Equation 13:

$$m = \frac{1}{1+e^{0}} \approx 0.5 \tag{13}$$

If there is a 2-bit difference ("01"), the hidden neuron has a value given by Equation 14:

$$m = \frac{1}{1+e^{L}} \approx 0 \tag{14}$$

In general, if an input string differs from the string, *s*, by 0 bits, 1 bit, 2 bits, 3 bits, etc., the hidden neuron identifies the string with values given in Equation 15:

$$m = 1, 0.5, 0, 0, \ldots \tag{15}$$

This hidden neuron can clearly identify, and only identify, one string or one subset, *s*. Consider this next example. Let a given target function hold the four inputs given in Table 4.

Table 4. A sample binary function with four inputs.

x_0	x_1	x_2	x_3	y
0	0	1	1	1
1	0	0	1	1

The rest of the rows in Table 4 all have y(x) = 0. The strings are y = {0011, 1001}, and the set of subsets is y = {{2,3}, {0,3}}. Table 5 gives the weights and biases of the hidden neurons (a_0, a_1, b in Equation 10) in two hidden trees. Each row specifies all parameters in a hidden tree. Column 1 is the input string. Column 2 (m_0) and Column 3 (m_1) are hidden neurons in the first hidden layer. Column 4 (m_2) is the hidden neuron in the last hidden layer.

Here, m_0 and m_1 are hidden neurons in the first hidden layer and m_2 is the one in the last hidden layer. Figure 4 shows that there is one tree for each string/subset.

Table 5. The weights and biases of the hidden neurons for two strings.

String	m_0	m_1	m_2
0011	(-L,-L,L)	(L,L,-L)	(L,L,-L)
1001	(-L,L,0)	(L,-L,0)	(L,L,-L)

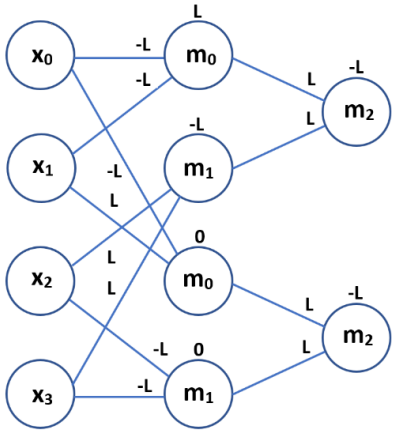


Figure 4. An example of two hidden trees for two strings: 0011 and 1001.

A target function is written in terms of a set of strings. For each string in the target function, there is one hidden tree that can identify it and only it. In this example, there are four inputs: x_0, x_1, x_2, and x_3 and the strings in a target function are y = {0011, 1001}. Here, m_0 and m_1 are hidden neurons in the first hidden layer and m_2 is in the last hidden layer. The output layer is omitted in this figure. The connection weight is written next to the edges and the bias is written on top of the hidden neurons. In neuron-based computing, connection weights are members of neurons rather than members of the connection matrix; so, whenever possible, the weights are drawn closer to its owners. A target function is written in terms of a set of subsets. For each subset in the target function, there is one hidden tree that can identify it and only it. The last hidden layer has one neuron for each subset, so the neural network can implement any target function.

Why locally Connected? A Time and Space Complexity Analysis

Time complexity measures how the running time of an algorithm grows as the size of its input increases. Space complexity measures the amount of memory or storage required by an algorithm relative to the size of its input. The implicit assumption here is that the comparison between a fully connected network and locally connected network is based on the fact that

the same target function can be identified by both. Let *d* be the number of input neurons; let h = {s_0, s_1, …, } be a target function; and, let |h| be the number of strings in set *h*. In the fully connected network, there are *d* neurons from the input layer, |h| neurons from the hidden layer, and one neuron from the output layer for a total of d + |h| + 1 neurons. The hidden layer has d*|h| connections and the output layer has |h| connections. For one pass of training, the time and space complexities are T = O (d*|h|) and S = O (d*|h|).

For bilocal networks, there are *d* neurons from the input layer, |h| * (d-1) neurons from the |h| hidden trees, and one neuron from the output layer for a total of d + |h| *(d-1) + 1 neurons. The number of hidden neurons is significantly higher, which is increased by a factor of O(d), from |h| to |h| * (d-1). There are also |h| binary trees, where each tree has 2 * d – 2 connections. The hidden layer has (2 * d – 2) * |h| connections, and the output layer has |h| connections. The number of connection weights is roughly doubled. This trade-off has the potential of improving time complexity at a minor cost of more neurons and connection weights. The space complexity is primarily determined by connection weights, not by the number of neurons, so the space complexity does not increase when the number of neurons is increased by a factor of O(d). Doubling the weights will also not change the space complexity, which measures the order of magnitude, and a constant of 2 will not change the space complexity. For one pass of training, both the time and space complexities are T' = O (d*|h|) and S' = O (d*|h|).

From the time and space complexity analyses, there are no advantages for the locally connected network; however, this is not true for the following reasons. First, the DNN itself attempts to localize the network by dividing the network into many layers; the deeper the network, the more locally connected the network will become. Second, the von Neumann bottleneck, which refers to the limit of computing systems that stems from the separation of the central processing unit (CPU) and RAM, is another problem. Training of large networks demands substantial hardware because:

1) Parameters: the scale of these models is immense and the memory requirements to store and process these parameters are significant, prompting the transition from CPU to GPUs and from GPU to IPU, TPU, and NPU (Brown et al., 2020). The computational complexity is primarily driven by the extensive matrix multiplications and gradient descent calculations involved in backpropagation, which require multiple passes through the entire network (Goodfellow, Bengio, & Courville, 2016).
2) Training data: training on vast datasets requires not only significant storage but also powerful computational resources to handle the iterative processes involved in training (Devlin, Chang, Lee & Toutanova, 2018).
3) Training: high throughput for data processing necessitates advanced storage systems and network infrastructure to efficiently feed data to the model. The distributed nature of training across multiple GPUs/TPUs adds further complexity (Rajbhandari, Rasley, Ruwase & He, 2020).
4) Power: high energy consumption can also be a problem (Jouppi et al., 2017). Simply speaking, it is impossible for the cache memory to hold so much data, so most of the time and power are consumed by moving data. It can be significantly helpful if the computations are basically local. This reduction of thrashing will not reduce the accuracy of the computation.

It is essential to emphasize the significance of incorporating locality in neural networks and its implications for computation efficiency. Using the earlier example in this section, in a fully connected network, there are *d* neurons from the input layer and |h| neurons from the hidden layer; the hidden layer has d*|h| connections. For one pass of training, the time and space complexities are T = O (d*|h|) and S = O (d*|h|). When both d and |h| are very large, the connection matrix (d x |h| dimension) is very large, and only a small portion of this matrix can be held in RAM. To complete a matrix multiplication, a portion of a large matrix is loaded into RAM, then removed from RAM to make room, only to find that it will need to be reloaded again. Increasingly, computation times are spent on moving data from one place to another. Assuming the same matrix will need to be reloaded R times on average, the mathematical time complexity of T = O (d*|h|), which assumes unlimited RAM, is actually T = O (d*|h|*R), where R is the average number of reloads for a large matrix. R is 1 only if the memory is as large as d*|h|, which is simply not the case for large matrices.

For bilocal networks, the number of hidden neurons is significantly higher, which is increased by a factor of O(d); however, the number of connection weights is roughly the same order of magnitude. It is the connection weights that determine the time and space complexities. The number of connection weights is roughly doubled, but there is no large matrix here so the data does not need to be loaded and unloaded over and over again. Why is there no connection matrix in a neuron-based computation? The connection weights are members of neurons. There are two computations: forward computations of neuron values and backward computations of weight updates.

1) When neuron values in the first hidden layer are calculated, they can be calculated one neuron at a time; this is because all weights of a neuron are properties of this neuron. To update a neuron value, the members of this neuron alone are enough to complete the neuron value calculation. When all of the neurons are updated in the first hidden layer, the process can be repeated for the second hidden layer, again one neuron at a time. Note that there is no matrix.
2) Similarly, when new weights are calculated, they can be calculated in such a way that only one neuron is used at a time; this is because the weight update training related to one neuron is based on all of the weights connected to this neuron in a backward direction. When

all of the neurons in the output layer are processed, one can repeat the process for the last hidden layer, again one neuron at a time, gradually moving backward. Note that there is again no matrix. For example, one can compute the responsibility of a neuron based on all the weights connected to this neuron in a backward direction. In both cases, only one-dimensional arrays are used because only one neuron is processed at a time and these arrays will be loaded into RAM only once. For one pass of training, the time complexity is T' = O (d*|h|), which can be significantly faster than T = O (d*|h|*R), in the case of fully connected neural networks, where R is the average number of reloads for a large matrix.

Incorporating locality in neural networks can increase computation efficiency by a factor of R. It is this factor of R that opens a discussion for exploring an approach of locally connected neural networks as an alternative to globally connected models. OpenAI's GPT-3, the architecture underlying ChatGPT-3, is one of the largest and most sophisticated language models developed with known size (Brown et al., 2020). The largest GPT-3 model, often referred to as GPT-3 175B, has 96 layers (transformer blocks) and 175 billion parameters. Each layer has an Attention block and a Feedforward Network.

The attention block has four 12,288 x 12,288 matrices, where three of the matrices will multiply (Vaswani et al., 2017). The Feedforward Network (FFN) has two 12,288 x 49,152 matrices. Together, these matrices give the majority of the 175 billion parameters. The number of parameters in GPT-4 is not officially disclosed by OpenAI, but it is expected that the cost of training ChatGPT-4 is an order of magnitude higher than ChatGPT-3 and the cost of training ChatGPT-5 will be an order of magnitude higher than ChatGPT-4 (Wall Street Journal, 2024). Therefore, it is important to increase computation efficiency.

In a locally connected network, the basic computation unit is a neuron rather than a connection matrix. In the extreme case of a bilocal network, a neuron value, a few weights, and a bias form the foundation of computation, irrespective of how large the network is. This is in contrast to the connection-matrix-based computation unit that grows with the network size. As a reference, biological neural networks are locally connected, where data movement is minimum and the number of neurons is large. The biggest difference between fully connected networks and bilocal networks is that one uses matrix-based computation and the other uses neuron-based computation; one uses matrices as a computation unit and the other uses neurons as computation units. As a reference, the neuroscience textbook by Kandel, Schwartz, and Jessell (2013) states that individual neurons in the human brain typically form between 1000 and 10,000 synaptic connections. The biological neural net has two features: it has a large number of neurons and is locally connected.

Discussion

Earlier, the authors made two assumptions for easy discussion: bilocal and 2^K input neurons. Now these assumptions will be removed.

Arbitrary number of input neurons:
To move from 2^K to an arbitrary number, the process is standard and well-known, such as binary search and merge sort. For example, let the input layer have 11 neurons:

[0,1,2,3,4,5,6,7,8,9,10]

Following the binary search or merge sort process, the division for an integer interval [a, b] is [a, m] and [m+1, b], where m = (a + b)/2 is an integer division. The division process then is:

[0,1,2,3,4,5,6,7,8,9,10]
[0,1,2,3,4,5], [6,7,8,9,10]
[0,1,2], [3,4,5], [6,7,8], [9,10]
[0,1], [2], [3,4], [5], [6,7], [8], [9,10]

Now there are some singleton neurons left in the input layer. A single neuron can be identified by a hidden neuron using the same rule noted previously.

N-ary tree:
N-ary tree is a tree in which a node can have at most N children. Binary trees are specific cases where N = 2. The binary connections are merely for easy discussion. By using an N-ary tree, all the restrictions that were imposed, for the sake of easy discussions, are removed. The rules allow one hidden neuron to identify arbitrary numbers of bits; therefore, all of the rules apply to the N-ary trees, which is: let *s* be a subset given by Equations 8 and 9, and assume that a hidden tree will identify *s*; the neurons in the first hidden layer then have weights and biases as follows:

set weight = L, for input neurons $\{j_0, j_1, j_2, \ldots\}$
set weight = – L, for all other input neurons
set bias = $-(|s| - 1) \cdot L$

Neurons in the rest of hidden layers have weights and biases determined by the above rule for identification of patterns: "11…1".

Universal Approximator with Two and Three Hidden Layers

For a given input number, *d*, and a given number of layers, there are numerous constructions, where the number of neurons in the hidden layers depends on the construction. Figure 5 gives an example of a single hidden tree for d = 8, two hidden layers, and the maximum localization construction (the number of edges is maximum). The figure shows the input layer, the first hidden layer, and the last hidden layer. The

output layer is omitted. Figure 6 gives an example of a single hidden tree for d = 8, two hidden layers, and the minimum localization construction (the number of edges is minimum). Again, the figure shows the input layer, the first hidden layer, and the last hidden layer. The output layer is omitted.

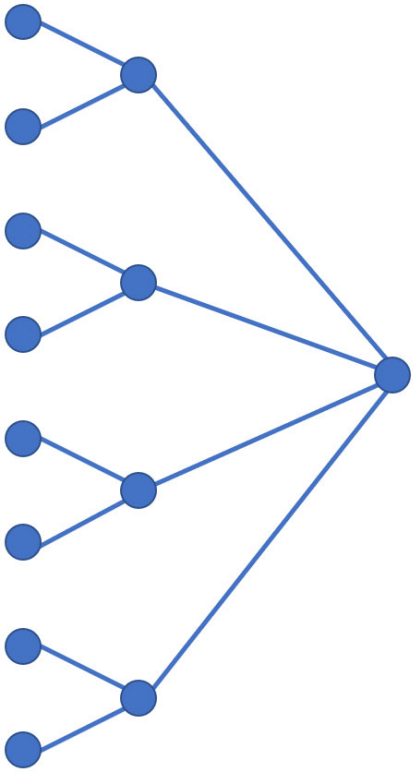


Figure 5. An example of maximum localization construction.

As *d* grows larger, the diagram gets harder to read, so a new notation will be introduced:

- Let the input neurons be labeled by "Input Layer: 0, 1, 2, …, 15";
- Let the neurons in the first hidden layer be labeled by "First Hidden Layer: 0, 1, 2, ….";
- Let the neurons in the second hidden layer be labeled by "Second Hidden Layer: 0, 1, 2, …";
- Let the neurons in the last hidden layer be labeled by "Last Hidden Layer: 0, 1, 2, …"; and,
- Let "[...]" be used to group neurons together to be identified by a neuron in the next layer.

For example, Input layer: [0,1] [2,3] means input neurons 0 and 1 will be identified by neuron 0 in the first hidden layer and input neurons 2 and 3 will be identified by neuron 1 in the first hidden layer. Under this notation, Figure 5 can be rewritten as:

Input layer: [0,1] [2,3] [4,5] [6,7]
First hidden layer: [0,1,2,3]
Last hidden layer: [0]

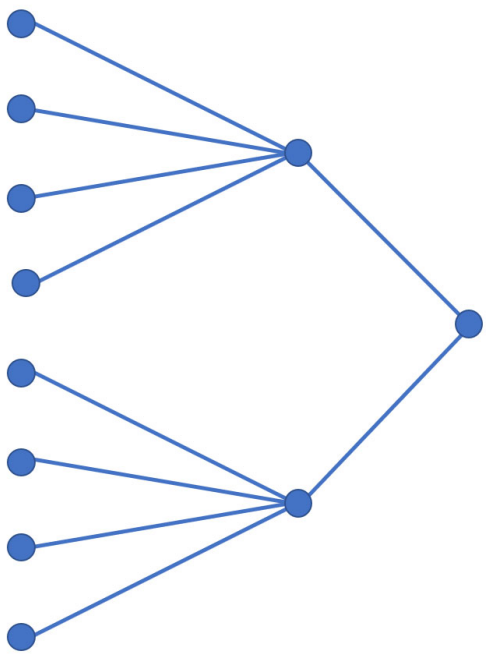


Figure 6. An example of minimum localization construction.

Figure 6 can be rewritten as:

Input layer: [0,1,2,3] [4,5,6,7]
First hidden layer: [0,1]
Last hidden layer: [0]

The following example will be more interesting, which is d = 16 and two hidden layers. In this example, let d = 16 and let a network have two hidden layers. The maximum localization construction looks like this:

Input layer: [0,1] [2,3] [4,5] [6,7][8,9][10,11][12,13][14,15]
First hidden layer: [0,1,2,3,4,5,6,7]
Last hidden layer: [0]

where,

- input neurons [0,1] are identified by neuron 0 of the first hidden layer,
- input neurons [2,3] are identified by neuron 1 of the first hidden layer,
- …, and
- first-hidden-layer neurons, [0, …, 7], are identified by neuron 0 of the last hidden layer.

The minimum localization looks like this:

Input layer: [0,1,2,3,4,5,6,7] [8,9,10,11,12,13,14,15]
First hidden layer: [0,1]
Last hidden layer: [0]

The intermediate localization looks like this:

Input layer: [0,1,2,3] [4,5,6,7] [8,9,10,11] [12,13,14,15]
First hidden layer: [0,1,2,3]
Last hidden layer: [0]

Clearly, there are many other constructions. As a comparison, with two hidden layers and minimum localization, it basically divides the original fully connected network into two networks, which reduces the weight connection matrix. With two hidden layers and maximum localization, it basically reduces the size of the original fully connected network by half by taking the average of two inputs and combining it into one input, which again reduces the connection matrix. In either case, it is a small deviation from the original network. As more and more layers are added, the difference between fully connected networks and locally connected networks will get bigger and bigger; eventually, it will transit from a matrix-based computation to a neuron-based computation. Universal approximators with three hidden layers can be constructed in a similar way.

Conclusions

In earlier work by the authors, they showed that an arbitrary binary target function can be effectively rewritten in terms of a set of strings, or a set of subsets, and that a single hidden neuron can identify and only identify a single string or a single subset; therefore, an arbitrary binary target function can be effectively rewritten in the form of a neural network with one hidden layer, thus proving that deep neural networks can effectively implement any target mappings. In this paper, the authors imposed locality on the neural network and showed that an arbitrary binary target function can be effectively rewritten in the form of a locally connected DNN, which can have many hidden layers. When locality is imposed on the network, the basic computation unit can be shifted to neurons rather than connection matrices. Continuous loading of batches of data from storage into memory to processing units can be significantly reduced. By imposing locality, the computation power of the DNN is not decreased, but it can reduce thrashing, thus significantly increasing computation speed.

Acknowledgments

This work was supported by the Department of Energy Minority Serving Institution Partnership Program (EM-MSIPP) managed by the Savannah River National Laboratory under BSRA contract TOA Number: 0000663608 and National Science Foundation under Award Number 2348805. The authors would like to thank Gina Porter for proofreading this paper.

References

Amari, S., Kurata, K., & Nagaoka, H. (1992). Information Geometry of Boltzmann Machine. *IEEE Trans., Neural Network, 3*(2), 260-271.

Anthropic. (2023). Claude: A Language Model for Conversational AI. https://www.anthropic.com/index/claude

Bengio, Y. (2009). Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning, 2*(1), 1-127. http://dx.doi.org/10.1561/2200000006

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 35* (8), 1798-1828. doi:10.1109/tpami.2013.50 https://ieeexplore.ieee.org/document/6472238

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P. ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877-1901.

Byrne, W. (1992). Alternating Minimization and Boltzmann Machine Learning. *IEEE Trans., Neural Network, 3*(4), 612-620.

Cheng, X., Li, Y., & Lu, J. (2019). Butterfly-Net: Optimal Function Representation Based on Convolutional Neural Networks. https://arxiv.org/pdf/1805.07451

Ciresan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3642-3649. doi:10.1109/cvpr.2012.6248110 https://ieeexplore.ieee.org/document/6248110

Coursera. (2017). Coursera, Your Course to Success. https://www.coursera.org/

Cybenko, G. (1989). Approximation by Superposition of a sigmoid function. *Mathematics of Control, Signals, and System, 2*, 303-314.

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/arXiv.1810.04805

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural networks. *Advances in Neural Information Processing Systems, 28*, 1135-1143.

Hinton, G. E., Osindero, S., & Teh, Y. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation, 18*, 1527-1554.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer Feedforward Networks are Universal Approximators. *Neural Networks, 2*, 359-366.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T. ... Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. https://doi.org/10.48550/arXiv.1704.04861

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R. ... Yoon, D. (2017). In-datacenter performance

analysis of a tensor processing unit. *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 1-12.
Kandel, E. R., Schwartz, J. H., & Jessell, T. M. (2013). *Principles of Neural Science* (5th ed.). McGraw-Hill Education.
Kubat, M. (2015). *An Introduction to Machine Learning.* (1st ed.). Springer.
Le Roux, N., & Bengio, Y. (2008). Representational Power of Restricted Boltzmann Machines and Deep Belief Networks. *Neural Computation, 20*(6), 1631-1649.
LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324.
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature, 521*(7553), 436-444.
Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D. ... Wu, H. (2018). *Mixed precision training*. International Conference on Learning Representations. https://doi.org/10.48550/arXiv.1710.03740
Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems, 25*, 1097-1105.
Liu, Y. (1993). Image Compression Using Boltzmann Machines. *Proc. SPIE, 2032*, 103-117.
Liu, Y. (1995). Boltzmann Machine for Image Block Coding. *Proc. SPIE, 2424*, 434-447.
Liu, Y. (1997). Character and Image Recognition and Image Retrieval Using the Boltzmann Machine. *Proc. SPIE*, 3077, 706-715.
Liu, Y. (2002). Attrasoft Image Retrieval. US Patent, *7,773,800*. http://www.google.com/patents/US7773800
Liu, Y., & Wang, S. H. (2018). Completeness Problem of the Deep Neural Networks. *American Journal of Computational Mathematics, 8*, 184-196. https://doi.org/10.4236/ajcm.2018.82014
Liu, Y. (2018a). Linear Neurons and Their Learning Algorithms. *Journal of Computer Science and Information Technology, 6*(2), 1-14.
Liu, Y. (2018b). Square Neurons, Power Neurons, and Their Learning Algorithms. *American Journal of Computational Mathematics, 8*, 296-313. https://doi.org/10.4236/ajcm.2018.84024
Liu, Y., & Yousuf, A. (2020). Deep Neural Network and Universal Approximators. *International Journal of Modern Engineering, 20*(2), 45-50. https://ijme.us/issues/spring2020/X__IJME%20spring%202020%20v20%20n2.pdf#page=47
OpenAI. (2023). ChatGPT: Language Model for Dialogue Applications. https://openai.com/chatgpt
Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020). *ZeRO: Memory Optimization Towards Training A Trillion Parameter Models*. https://doi.org/10.48550/arXiv.1910.02054
Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks, 61*, 85-117. https://doi.org/10.1016/j.neunet.2014.09.003
Tan, M., & Le, Q. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. *Proceedings of the 36th International Conference on Machine Learning*, 6105-6114.
Theano. (2017). Theano 1.0. https://en.wikipedia.org/wiki/Theano_(software)
Tensorflow, (2017). Tensorflow. https://www.tensorflow.org
Torch, (2019). Torch, A scientific computing framework for LuaJIT. http://torch.ch/
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N. … Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 5998-6008.
Wall Street Journal. (2024). Eric Schmidt Walks Back Claim Google Is Behind on AI Because of Remote Work. https://www.wsj.com/tech/ai/google-eric-schmidt-ai-remote-work-stanford-f92f4ca5

Biographies

YING LIU is an Associate Professor of Computer Science Technology at Savannah State University. He received his master's degree and PhD in physics from Carnegie-Mellon University, and a Master's in Computer Science degree from the University of South Carolina. Dr. Liu has multiple Microsoft Certifications, including MCSE (Microsoft Certified System Engineer) and MCDBA (Database Administrator). He has published over 60 research papers, holds one patent, and over 30 software copyrights. Dr. Liu has extensive experience in software research and development in image recognition. Dr. Liu may be reached at liuy@savannahstate.edu

MAJID BAGHERI is an Assistant Professor of Civil Engineering Technology at Savannah State University. He received his PhD in Civil Engineering (environmental focus) from Missouri University of Science and Technology. Dr. Bagheri's expertise is in the modeling of environmental systems and environmental remediation using novel technologies such as artificial intelligence (AI) and machine learning (ML). He has developed several AI and ML models to improve the efficiency of treatment systems and assess the fate of environmental contaminants. Dr. Bagheri may be reached at bagherim@savannahstate.edu

ANTONIO VELAZQUEZ is an assistant professor at Savannah State University. He earned his BS in Civil Engineering (Structures and Construction) from Metropolitan Autonomous University (UAM-Mexico), a MEng in Structures from National Autonomous University of Mexico, a MSc in Computer Systems from National Polytechnic Institute (Mexico), a MSc in Wind Engineering from Northeastern University, a MSc in Fluid Dynamics (Hurricane Engineering) from Florida Institute of Technology, a PhD in Engineering Mechanics/Structures/MechEng from Michigan Technological University, and a post-doc in Machine-learning-based Engineering Education from Jackson State University. His research interests include

broad fields of engineering and computational mechanics. He is the author of several homemade CAD-like engineering software platforms developed at a commercial quality level. Dr. Velazquez has published research articles in international journals, conference proceedings, and workshops such as IWSHM, Journal of Sound and Vibration, Journal of Intelligent Material Systems and Structures, Journal of Computers (IEEE), and Journal of Engineering Structures. Dr. Velazquez may be reached at velazqueza@savannahstate.edu

ASAD YOUSUF is the department chairman for the Engineering Technology Department at Savannah State University. He received his BS in Electrical Engineering from NED University, MS in Electrical Engineering from the University of Cincinnati with emphasis on computer systems, and his doctorate from the University of Georgia. He has published several papers in technical journals and conference proceedings. He has years of experience in managing federally funded grants, having received over $2.5M in grant awards in the last few years. Dr. Yousuf may be reached at yousufa@savannahstate.edu

INSTRUCTIONS FOR AUTHORS: MANUSCRIPT FORMATTING REQUIREMENTS

The INTERNATIONAL JOURNAL OF MODERN ENGINEERING is an online/print publication designed for Engineering, Engineering Technology, and Industrial Technology professionals. All submissions to this journal, submission of manuscripts, peer-reviews of submitted documents, requested editing changes, notification of acceptance or rejection, and final publication of accepted manuscripts will be handled electronically. The only exception is the submission of separate high-quality image files that are too large to send electronically.

All manuscript submissions must be prepared in Microsoft Word (.doc or .docx) and contain all figures, images and/or pictures embedded where you want them and appropriately captioned. Also included here is a summary of the formatting instructions. You should, however, review the sample Word document on our website (http://ijme.us/formatting_guidelines/) for details on how to correctly format your manuscript. The editorial staff reserves the right to edit and reformat any submitted document in order to meet publication standards of the journal.

The references included in the References section of your manuscript must follow APA-formatting guidelines. In order to help you, the sample Word document also includes numerous examples of how to format a variety of scenarios. Keep in mind that an incorrectly formatted manuscript will be returned to you, a delay that may cause it (if accepted) to be moved to a subsequent issue of the journal.

1. **Word Document Page Setup:** Two columns with ¼" spacing between columns; top of page = ¾"; bottom of page = 1" (from the top of the footer to bottom of page); left margin = ¾"; right margin = ¾".

2. **Paper Title:** Centered at the top of the first page with a 22-point Times New Roman (Bold), small-caps font.

3. **Page Breaks:** Do not use page breaks.

4. **Figures, Tables, and Equations:** All figures, tables, and equations must be placed immediately after the first paragraph in which they are introduced. And, each must be introduced. For example: "Figure 1 shows the operation of supercapacitors." "The speed of light can be determined using Equation 4:"

5. **More on Tables and Figures:** Center table captions above each table; center figure captions below each figure. Use 9-point Times New Roman (TNR) font. Italicize the words for table and figure, as well as their respective numbers; the remaining information in the caption is not italicized and followed by a period—e.g., "*Table 1*. Number of research universities in the state." or "*Figure 5*. Cross-sectional aerial map of the forested area."

6. **Figures with Multiple Images:** If any given figure includes multiple images, do NOT group them; they must be placed individually and have individual minor captions using, "(a)" "(b)" etc. Again, use 9-point TNR.

7. **Equations:** Each equation must be numbered, placed in numerical order within the document, and introduced—as noted in item #4.

8. **Tables, Graphs, and Flowcharts:** All tables, graphs, and flowcharts must be created directly in Word; tables must be enclosed on all sides. The use of color and/or highlighting is acceptable and encouraged, if it provides clarity for the reader.

9. **Textboxes:** Do not use text boxes anywhere in the document. For example, table/figure captions must be regular text and not attached in any way to their tables or images.

10. **Body Fonts:** Use 10-point TNR for body text throughout (1/8" paragraph indention); indent all new paragraphs as per the images shown below; do not use tabs anywhere in the document; 9-point TNR for author names/affiliations under the paper title; 16-point TNR for major section titles; 14-point TNR for minor section titles.

11. **Personal Pronouns:** Do not use personal pronouns (e.g., "we" "our" etc.).

12. **Section Numbering:** Do not use section numbering of any kind.

13. **Headers and Footers:** Do not use either.

14. **References in the Abstract:** Do NOT include any references in the Abstract.

15. **In-Text Referencing:** For the first occurrence of a given reference, list all authors—last names only—up to seven (7); if more than seven, use "et al." after the seventh author. For a second citation of the same reference—assuming that it has three or more authors—add "et al." after the third author. Again, see the *sample Word document* and the *formatting guide for references* for specifics.

16. **More on In-Text References:** If you include a reference on any table, figure, or equation that was not created or originally published by one or more authors on your manuscript, you may not republish it without the expressed, written consent of the publishing author(s). The same holds true for name-brand products.

17. **End-of-Document References Section:** List all references in alphabetical order using the last name of the first author—last name first, followed by a comma and the author's initials. Do not use retrieval dates for websites.

18. **Author Biographies:** Include biographies and current email addresses for each author at the end of the document.

19. **Page Limit:** Manuscripts should not be more than 15 pages (single-spaced, 2-column format, 10-point TNR font).

20. **Page Numbering:** Do not use page numbers.

21. **Publication Charges:** Manuscripts accepted for publication are subject to mandatory publication charges.

22. **Copyright Agreement:** A copyright transfer agreement form must be signed by all authors on a given manuscript and submitted by the corresponding author before that manuscript will be published. Two versions of the form will be sent with your manuscript's acceptance email.

    Only one form is required. Do not submit both forms!

    The form named "paper" must be hand-signed by each author. The other form, "electronic," does not require hand signatures and may be filled out by the corresponding author, as long as he/she receives written permission from all authors to have him/her sign on their behalf.

23. **Submissions:** All manuscripts and required files and forms must be submitted electronically to Dr. Philip D. Weinsier, manuscript editor, at philipw@bgsu.edu.

24. **Published Deadlines:** Manuscripts may be submitted at any time during the year, irrespective of published deadlines, and the editor will automatically have your manuscript reviewed for the next-available issue of the journal. Published deadlines are intended as "target" dates for submitting new manuscripts as well as revised documents. Assuming that all other submission conditions have been met, and that there is space available in the associated issue, your manuscript will be published in that issue if the submission process—including payment of publication fees—has been completed by the posted deadline for that issue.

    Missing a deadline generally only means that your manuscript may be held for a subsequent issue of the journal. However, conditions exist under which a given manuscript may be rejected. Always check with the editor to be sure. Also, if you do not complete the submission process (including all required revisions) within 12 months of the original submission of your manuscript, your manuscript may be rejected or it may have to begin the entire review process anew.

CUTTING EDGE JOURNAL OF RESEARCH AND INNOVATION IN ENGINEERING

Print ISSN: 2152-4157
Online ISSN: 2152-4165

www.ijeri.org

www.iajc.org

INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH AND INNOVATION

ABOUT IJERI:

- IJERI is the second official journal of the International Association of Journals and Conferences (IAJC).
- IJERI is a high-quality, independent journal steered by a distinguished board of directors and supported by an international review board representing many well-known universities, colleges, and corporations in the U.S. and abroad.
- IJERI has an impact factor of **1.58**, placing it among an elite group of most-cited engineering journals worldwide.

OTHER IAJC JOURNALS:

- The International Journal of Modern Engineering (IJME) For more information visit www.ijme.us
- The Technology Interface International Journal (TIIJ) For more information visit www.tiij.org

IJERI SUBMISSIONS:

- Manuscripts should be sent electronically to the manuscript editor, Dr. Philip Weinsier, at philipw@bgsu.edu.

  For submission guidelines visit www.ijeri.org/submissions

TO JOIN THE REVIEW BOARD:

- Contact the chair of the International Review Board, Dr. Philip Weinsier, at philipw@bgsu.edu.

  For more information visit www.ijeri.org/editorial

INDEXING ORGANIZATIONS:

- IJERI is currently indexed by 16 agencies. For a complete listing, please visit us at www.ijeri.org.

Contact us:

Mark Rajai, Ph.D.

Editor-in-Chief
California State University-Northridge
College of Engineering and Computer Science
Room: JD 4510
Northridge, CA 91330
Office: (818) 677-5003
Email: mrajai@csun.edu

www.tiij.org

www.ijme.us

THE LEADING JOURNAL OF ENGINEERING, APPLIED SCIENCE AND TECHNOLOGY

The latest impact factor (IF) calculation (Google Scholar method) for IJME of 3.0 moves it even higher in its march towards the top 10 engineering journals.

IJME IS THE OFFICAL AND FLAGSHIP JOURNAL OF THE INTERNATIONAL ASSOCATION OF JOURNALS AND CONFERENCE (IAJC)
www.iajc.org

The International Journal of Modern Engineering (IJME) is a highly-selective, peer-reviewed journal covering topics that appeal to a broad readership of various branches of engineering and related technologies. IJME is steered by the IAJC distinguished board of directors and is supported by an international review board consisting of prominent individuals representing many well-known universities, colleges, and corporations in the United States and abroad.

IJME Contact Information

General questions or inquiries about sponsorship of the journal should be directed to:

Mark Rajai, Ph.D.
Editor-in-Chief
Office: (818) 677-5003
Email: editor@ijme.us
Department of Manufacturing Systems Engineering & Management
California State University-Northridge
18111 Nordhoff St.
Northridge, CA 91330