Space vector modulation (SVM) is a technique used for generating alternating current waveforms to control pulse width modulation signals (PWM). It provides better results of PWM signals compared to other techniques. CORDIC algorithm calculates hyperbolic and trigonometric functions of sine, cosine, magnitude and phase using bit shift, addition and multiplication operations.

This thesis implements SVM with Arduino microcontroller using CORDIC algorithm. This algorithm is used to calculate the PWM timing signals which are used to control the motor. Comparison of the time taken to calculate sinusoidal signal using Arduino and CORDIC algorithm was also done.
CONTROLLING AC MOTOR USING ARDUINO MICROCONTROLLER

BY

NITHESH REDDY NANNURI

©2014 Nithesh Reddy Nannuri

A THESIS SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

Thesis Director:
Dr. Donald S Zinger
ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Donald S. Zinger for his continuous support and guidance in this thesis work as well as throughout my graduate study.

I would like to thank Dr. Martin Kocanda and Dr. Peng-Yung Woo for serving as members of my thesis committee.

I would like to thank my family for their unconditional love, continuous support, enduring patience and inspiring words. Finally, I would like to thank my friends and everyone who has directly or indirectly helped me for their cooperation in completing the thesis.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>List</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
</tbody>
</table>

## Chapter

1. INTRODUCTION
   1.1 Scalar control
   1.2 Vector control
   1.3 Arduino UNO
   1.4 Space vector modulation
   1.5 CORDIC
   1.6 Thesis Organization

2. ARDUINO UNO
   2.1 Summary of Arduino UNO
   2.2 Power supply
   2.3 Input and output
   2.4 Communication
   2.5 Programming and reset
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6 Atmega328</td>
<td>11</td>
</tr>
<tr>
<td>2.7 Arduino UNO software</td>
<td>14</td>
</tr>
<tr>
<td>3. SPACE VECTOR MODULATION</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Switching states</td>
<td>15</td>
</tr>
<tr>
<td>3.2 Voltage space vectors</td>
<td>18</td>
</tr>
<tr>
<td>3.3 Vector analysis of the inverter</td>
<td>20</td>
</tr>
<tr>
<td>3.4 Calculation of switching times</td>
<td>23</td>
</tr>
<tr>
<td>3.5 Switching sequence</td>
<td>25</td>
</tr>
<tr>
<td>4. CORDIC</td>
<td>27</td>
</tr>
<tr>
<td>4.1 Operation of CORDIC in rotating mode</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Implementing CORDIC</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Steps for implementing CORDIC</td>
<td>34</td>
</tr>
<tr>
<td>5. RESULTS</td>
<td>36</td>
</tr>
<tr>
<td>6. CONCLUSION</td>
<td>39</td>
</tr>
<tr>
<td>6.1 Future scope of work</td>
<td>39</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>40</td>
</tr>
<tr>
<td>APPENDIX: CODE LISTING</td>
<td>42</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Summary of Arduino UNO</td>
<td>8</td>
</tr>
<tr>
<td>2. Relation between space vectors and switching states</td>
<td>22</td>
</tr>
<tr>
<td>3. Seven-segment switching sequence</td>
<td>26</td>
</tr>
<tr>
<td>4. Lookup table for the values of $\tan^{-1}(2^{-i})$</td>
<td>32</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Arduino UNO microcontroller board</td>
<td>7</td>
</tr>
<tr>
<td>2.</td>
<td>Block diagram of Atmega328</td>
<td>13</td>
</tr>
<tr>
<td>3.</td>
<td>Selecting Arduino UNO board</td>
<td>14</td>
</tr>
<tr>
<td>4.</td>
<td>A voltage source inverter</td>
<td>16</td>
</tr>
<tr>
<td>5.</td>
<td>Eight switching states of three-phase inverter</td>
<td>17</td>
</tr>
<tr>
<td>6.</td>
<td>Representation of $V/I (PNN)$ in $\alpha, \beta$ plane</td>
<td>18</td>
</tr>
<tr>
<td>7.</td>
<td>Non-zero voltage vectors in $\alpha, \beta$ plane</td>
<td>19</td>
</tr>
<tr>
<td>8.</td>
<td>Representation of zero-vector in $\alpha, \beta$ plane</td>
<td>20</td>
</tr>
<tr>
<td>9.</td>
<td>Generating vector $\mathbf{V}$ from $\mathbf{V}_1, \mathbf{V}_2$ and $\mathbf{V}_7$</td>
<td>23</td>
</tr>
<tr>
<td>10.</td>
<td>Seven-segment switching sequence in sector 1</td>
<td>25</td>
</tr>
<tr>
<td>11.</td>
<td>Perfect rotation of a vector in a plane</td>
<td>28</td>
</tr>
<tr>
<td>12.</td>
<td>General implementation of CORDIC</td>
<td>31</td>
</tr>
<tr>
<td>13.</td>
<td>Flow chart of implementation of CORDIC</td>
<td>35</td>
</tr>
<tr>
<td>14.</td>
<td>CORDIC generated for $30^0$</td>
<td>37</td>
</tr>
<tr>
<td>15.</td>
<td>PWM waveforms for frequency 50 Hz using CORDIC</td>
<td>38</td>
</tr>
</tbody>
</table>
CHAPTER-1

INTRODUCTION

Three-phase induction motors are widely used in many applications and several methods are available to control the speed and torque of the motor. By varying the load, speed of the motors can be controlled and energy can be saved. There are a few techniques involved in controlling the speed of the motor. They are classified as scalar control and vector control. [1]

1.1 Scalar Control:

In scalar control method speed can be varied by changing the supply frequency which results in change of impedances. This change of impedances might increase the current or decrease it. If the current is small, torque of the motor decreases. Frequency and impedances are directly proportional and if frequency decreases, coils can be burned or saturation can occur in the iron of coils. To avoid this both voltage and frequency are varied at the same time, compensating the disadvantages of changing frequency alone.

1.2 Vector Control:

Field-oriented control is one of the methods of vector control. Direct torque control (DTC) and direct self-control methods also work with vectors. The field-oriented control works with the principle of rotating vectors in a complex coordinate system. Here magnitude and phase are controlled with change in current. With this field components are uncoupled,
establishing two independent currents: flux-producing current and torque-producing current which can be used to control current and flux independently. Induction motor loses its complexity and high performance can be realized [2] by maintaining $90^0$ electrical angles between uncoupled control currents.

This control technique can be carried out by applying system and coordinate transformations to the basic equations of the motor. Alternating and sinusoidal quantities become non alternating quantities. Magnitude and phase of supply voltage or current can be modified by back-transformation with three phase quantities.

Scalar control method is cheap and simple compared with field-oriented control. It controls magnitude of voltages and frequency instead of controlling phase and magnitude of currents. On the other hand, field-oriented control method controls the current and operates with fast responses.

### 1.3 Arduino UNO:

Arduino UNO is a microcontroller based on ATMEGA328. It has 14 digital input/output pins out of which six pins can be used as PWM outputs, six analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support microcontroller. Simply connect it to a computer via USB or else connect directly with a battery to get started. [3]

Arduino UNO can be programmed with Arduino software. The open-source Arduino environment makes it easy to write code and upload it to I/O board through USB. It runs on all
platforms like Windows, Mac OSX and LINUX. CORDIC algorithm and Space vector modulation are implemented by using Arduino microcontroller. The switching times are calculated by implementing the code through Arduino.

1.4 Space Vector Modulation (SVM):

Space vector modulation is used for controlling pulse width modulation (PWM). This is the one of the best methods available for PWM signals. It is used for the creation of AC waveforms. Usually SVM works well if all the three output voltages of the inverter on the motor are taken into account for generating switching patterns.

There are eight switching states for the inverter at any instant of time. These states are represented by stationary vectors, out of which six vectors are active vectors and the remaining two vectors are zero vectors. The output at any instant of time is given by the reference vector $V_{ref}$, which can be synthesized by three stationary vectors. The on and off times of the inverter switch are calculated and the switching sequence is chosen to minimize the number of switching.

1.5 CORDIC:

CORDIC abbreviates COordinate Rotation Digital Computer. It is an algorithm designed to compute trigonometric and hyperbolic functions using bit shift, addition operations by eliminating multiplication operations. This can be operated in vector mode and rotating mode. The angle whose cosine or sine has to be found is stored in an accumulator and each step is obtained by direction of rotation of the vector, based on the sign of residual angle.
The CORDIC equations in rotation mode are given by:

\[ x(i+1) = x(i) - d_i \cdot 2^{-i} \cdot y(i) \]
\[ y(i+1) = y(i) + d_i \cdot 2^{-i} \cdot x(i) \]
\[ z(i+1) = z(i) - d_i \cdot \tan^{-1}(2^{-i}) \]

where \( d_i = -1 \) if \( z(i) < 0 \), \(+1\) otherwise. [4]

‘\( x \)’ is started with the value of constant \( 1/K \) and is obtained by the product of cosine values. At each iteration ‘\( x \)’ and ‘\( y \)’ values are updated. ‘\( z \)’ values are updated to zero to make error as low as possible and at certain iteration ‘\( z \)’ value becomes zero. At the end, ‘\( x \)’ value gives cosine of the angle and ‘\( y \)’ value gives sine of that particular angle. The sign of ‘\( d \)’ depends on angle of rotation.

1.6 Thesis Organization:

In this thesis SVM is implemented by Arduino UNO microcontroller using CORDIC algorithm. The second chapter deals with the Arduino UNO board, discussing the digital pins, power supply and the microcontroller associated with it. The features of Arduino and its software are discussed in the second chapter.

The third chapter deals with the space vector modulation. The representation of reference vector, how the switching states are represented and the calculation of switching times are discussed there. The selection of switching sequence is important to reduce harmonic effects and the procedure is represented there.
The fourth chapter includes the calculation of sine and cosine of a particular angle using CORDIC algorithm eliminating the need for multiplication. The fifth chapter contains the results obtained from Arduino software and the waveforms associated with it. Conclusions and future work to extend this thesis are mentioned in the sixth chapter.
CHAPTER-2

ARDUINO UNO

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors and other actuators. The microcontroller on the board is programmed using the Arduino Programming Language and the Arduino development environment. Arduino projects can be stand-alone or they can communicate with software running on a computer. Arduino is an open-source electronics platform based on easy-to-use hardware and software. It is a single-board microcontroller (microcontroller built onto a single printed circuit board).

Arduino UNO is based on ATmega328. It has 14 digital input/output pins of which six can be used as PWM outputs. It has six analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller and is connected to the computer using USB cable or AC-DC adapter or battery to get started. [3] Figure [1] shows an Arduino UNO microcontroller board.
Figure 1: Arduino UNO microcontroller board [3]
### 2.1 Summary of Arduino UNO:

Table 1: Summary of Arduino UNO.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller</td>
<td>ATmega328</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>5V</td>
</tr>
<tr>
<td>Input voltage</td>
<td>7V - 12V</td>
</tr>
<tr>
<td>Output voltage (limits)</td>
<td>6V – 12V</td>
</tr>
<tr>
<td>Digital I/O pins</td>
<td>16 (6 PWM pins)</td>
</tr>
<tr>
<td>Analog input pins</td>
<td>6</td>
</tr>
<tr>
<td>DC current per I/O pin</td>
<td>40 mA</td>
</tr>
<tr>
<td>DC current for 3.3V pin</td>
<td>50 mA</td>
</tr>
<tr>
<td>Flash Memory</td>
<td>32 KB</td>
</tr>
<tr>
<td>SRAM</td>
<td>2 KB</td>
</tr>
<tr>
<td>EEPROM</td>
<td>1 KB</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16 MHz</td>
</tr>
</tbody>
</table>
2.2 Power Supply:

The Arduino can be powered via USB connection or external power supply. Power source is selected automatically. External power can come from AC-DC supply or battery. The board can be operated on an external supply of 6V-20V. The recommended voltage is 7V-12V. If the board is supplied with less than 7V, it will be unstable, and if it is supplied with more than 12V, the voltage regulator is overheated and may damage the board.

VIN: The board is supplied with 5V through USB or from an external battery.

5V: This pin outputs a regulated 5V from the regulator onto the board. Power can be supplied with DC jack (7V–12V), USB connector (5V), or the VIN pin on the board.

3.3V: A 3.3V volt supply is generated by on-board regulator.

GND: Ground pin.

IOREF: Provides the voltage reference with which microcontroller operates. Selects the appropriate power source or enables voltage translators on the output working with 5V or 3.3V.[3]

2.3 Input and Output:

Each pin in the board can be used as input or output pin using pinMode(), digitalWrite() and digitalRead() functions. Each pin is operated with 5V.
- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on; when the pin is LOW, it's off.
- The UNO has six analog inputs, labeled A0 through A5, each of which provides 10 bits of resolution. By default they measure from ground to 5 volts.
- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.
- AREF. Reference voltage for the analog inputs.
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board. [3]

2.4 Communication:

Arduino UNO has several ways of communication with computers, other Arduinos or from other microcontrollers. ATmega328 provides UART TTL serial communication, which is available on digital pins 0 (RX) and 1 (TX). Arduino software has serial monitor which allows
simple data to be sent to and from Arduino board. RX and TX LEDs will flash while data is being transferred via USB through the computer.

2.5 Programming and Reset:

Arduino UNO can be programmed with Arduino software. It comes with a boot loader that allows uploading the new code without any external hardware programmer. Arduino software uses Arduino C language which is similar to C++. Once the coding is over, it is uploaded to the board through USB. While uploading TX and RX, LEDs will flash. Choose the board as Arduino UNO and select serial port from serial port menu. This is likely to be COM3 or higher.

Arduino UNO has reset button in the board. When it is pressed, the board will reset. Instead of doing so, UNO is designed in a way that allows it to reset by software running on a connected computer.

2.6 ATMega328:

Atmega328 is an 8-bit microcontroller with 32x8 general purpose-working registers. It has an on-chip 2-cycle multiplier. High-endurance non-volatile memory segments include 32K bytes of in-system self-programmable flash memory program, 1 K bytes of EEPROM and 2K bytes of internal SRAM. The write/erase cycles are set to 10,000 Flash/100,000 EEPROM.

The peripheral features of Atmega328 are:

- Two 8-bit timer/counters with separate prescaler and compare Mode
- One 16-bit timer/counter with separate prescaler, compare Mode, and capture mode
- Real-time counter with separate oscillator
- Six PWM Channels
- 8-channel 10-bit ADC in TQFP and QFN/MLF package
- 6-channel 10-bit ADC in PDIP Package
- Programmable Serial USART
- Master/Slave SPI Serial Interface
- Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Interrupt and Wake-up on Pin Change

It has 23 programmable I/O lines. The operating voltage is in between 1.8V – 5.5 V and the temperature range is -40°C to 85°C.

Atmega328 is a low-power CMOS 8-bit microcontroller based on AVR-enhanced RISC architecture. Atmega328 achieves 1 MIPS per MHz by executing powerful instructions allowing the system designer to optimize power consumption versus processing speed. [5] See Figure 2.
Figure 2: Block diagram of Atmega328 [5]

The AVR core combines a rich instruction set with 32 general-purpose working registers. All the 32 registers are directly connected to the arithmetic logic unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The device is manufactured using Atmel’s high-density non-volatile memory technology. Atmega328 is a powerful microcontroller that provides a highly flexible and cost-effective solution to many embedded control applications. [5]
2.7 Arduino UNO Software:

Arduino UNO has its own software. It is similar to ‘C’ language. Code has to be written in the software provided and uploaded the code to the board which is connected to the computer using USB cable. When using development software, selection of Arduino UNO board and serial port is important as shown in Figure 3.

![Selecting Arduino UNO board](image)

**Figure 3:** Selecting Arduino UNO board
CHAPTER-3

SPACE VECTOR MODULATION

In recent years, Space vector modulation (SVM) is used for controlling three-phase PWM inverters. This is the one of the best available methods for PWM signals. It is an advanced control mechanism that generates three-phase AC voltages of the desired magnitude and frequency at the output of the inverter. To implement SVM a reference signal $V_{ref}$ is sampled with frequency $f_s$ ($f_s = 1/T_s$) [6]. For generating switching patterns the output voltages of the inverter are taken into account. The vectorial representation was first presented in the contributions of Park [7] and Kron [8].

3.1 Switching States:

The voltage source inverter is shown in the Figure 4. A voltage source inverter can have only eight switching states because the input lines are never shorted and the output current is always continuous. These eight switching states are shown in the Figure 5. These states are represented by stationary vectors, out of which six vectors are active vectors and the remaining two vectors are zero vectors.
Each of the terminals ‘A’, ‘B’ and ‘C’ has only two levels with respect to ‘N’, \(V_g\) and 0 where \(V_g\) is the DC bus voltage. Therefore the switching state for each terminal can be denoted by ‘P’ and ‘N’. Here state ‘P’ represents when upper switch is ON and the output voltage of the inverter becomes \(+V_g\). ‘N’ represents the state when lower switch is ON and the voltage will be zero. Each of the eight switching states can be therefore denoted with three terms consisting of either ‘P’ or ‘N’.
Figure 5: Eight switching states of three-phase inverter
3.2 Voltage Space Vectors:

Space vector modulation is expressed as vectors in two-dimensional plane. Considering topology $V1 (PNN)$ which is shown in the Figure 5, the line voltages $V_{AB}$, $V_{BC}$ and $V_{CA}$ are given by

$$V_{AB} = V_{g}$$

$$V_{BC} = 0$$ and

$$V_{CA} = -V_{g}$$

This can be represented in the Figure 6, where line voltages are displaced in $120^0$. The A, B and C are connected to either positive or negative DC rail. In PNN, phase A is connected to positive DC rail and phases B and C are connected to negative DC rail as shown in the Figure [6],[10]

Figure 6: Representation of $V1 (PNN)$ in $\alpha$, $\beta$ plane
Accordingly, similar non-zero vectors \((V1 – V6)\) can be shown in positions as shown in the Figure 7. The area enclosed by two adjacent vectors within the hexagon is defined as a sector and those six sectors are as shown below.

![Figure 7: Non-zero voltage vectors in \(\alpha, \beta\) plane](image)

The output line voltages generated by topologies \(V0\) (PPP) and \(V7\) (NNN) are as follows:

\[
V_{AB} = 0
\]
\[
V_{BC} = 0 \text{ and }
\]
\[
V_{CA} = 0
\]

These vectors are called zero vectors and hence the magnitude of these voltage vectors is zero. They assume their position at the origin in the \(\alpha, \beta\) plane as shown in Figure 8. [10]
3.3 Vector Analysis of the Inverter:

The switching states of the inverter are represented by six active vectors $V1 - V6$ and two zero vectors $V0 - V7$. The inverter in balanced condition is represented as

$$V_{AO}(t) + V_{BO}(t) + V_{CO}(t) = 0 \quad (3.1)$$

where $V_{AO}(t)$, $V_{BO}(t)$ and $V_{CO}(t)$ are load phase voltages. So, if any of the two voltages are given, the third one can be calculated by using the above equation [9].

The three phase voltages can be transformed into two phase variables in $\alpha$–$\beta$ plane as follows:

$$\begin{pmatrix} V_\alpha(t) \\ V_\beta(t) \end{pmatrix} = \frac{2}{3} \begin{pmatrix} 1 & 1/2 & 1/2 \\ 0 & \sqrt{3}/2 & \sqrt{3}/2 \end{pmatrix} \begin{pmatrix} V_{AO}(t) \\ V_{BO}(t) \\ V_{CO}(t) \end{pmatrix} \quad (3.2)$$
Instead of analyzing the whole system, one could analyze looking at each phase.

Any space vector can be expressed in terms of two phase voltages in $\alpha, \beta$ plane as

$$\vec{v}(t) = V_{\alpha}(t) + j*V_{\beta}(t)$$

(3.3)

Put equation (3.2) in (3.3):

$$\vec{v}(t) = \frac{2}{3} \left( V_{AO}(t) e^{j0} + V_{BO}(t) e^{j2\frac{\pi}{3}} + V_{CO}(t) e^{j4\frac{\pi}{3}} \right)$$

(3.4)

For the switching state $V2$ (PPN), generated voltages are:

$$V_{AO}(t) = \frac{V_g}{3}$$

$$V_{BO}(t) = \frac{V_g}{3}$$

$$V_{CO}(t) = -2*\frac{V_g}{3}$$

The load voltages can be obtained by substituting eq (3.4) in to eq (3.3) and denoted by $\vec{V}_2$:

$$\vec{V}_2 = \frac{2}{3} * V_g * e^{j\frac{2\pi}{3}}$$

(3.5)

Similarly, all six vectors can be obtained by the following equation [9]:

$$\vec{V}_n = \frac{2}{3} * V_g * e^{j\frac{(n-1)\pi}{3}} \text{ where } n= 1 \text{ to } 6$$

(3.6)

See Table 2.
Table 2: Relation between Space Vectors and Switching States [9]

<table>
<thead>
<tr>
<th>Space Vector</th>
<th>Switching State</th>
<th>On-state Switch</th>
<th>Vector Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vec{V}_0 )</td>
<td>( PPP )</td>
<td>( S_1,S_3,S_5 )</td>
<td>( \vec{V}_0 = 0 )</td>
</tr>
<tr>
<td>( \vec{V}_7 )</td>
<td>( NNN )</td>
<td>( S_4,S_6,S_2 )</td>
<td>( \vec{V}_7 = 0 )</td>
</tr>
<tr>
<td>( \vec{V}_1 )</td>
<td>( PNN )</td>
<td>( S_1,S_6,S_2 )</td>
<td>( \vec{V}_1 = \frac{2}{3} * V_g * e^{j0} )</td>
</tr>
<tr>
<td>( \vec{V}_2 )</td>
<td>( PPN )</td>
<td>( S_1,S_3,S_2 )</td>
<td>( \vec{V}_2 = \frac{2}{3} * V_g * e^{j\frac{\pi}{3}} )</td>
</tr>
<tr>
<td>( \vec{V}_3 )</td>
<td>( NPN )</td>
<td>( S_4,S_3,S_2 )</td>
<td>( \vec{V}_3 = \frac{2}{3} * V_g * e^{j2\frac{\pi}{3}} )</td>
</tr>
<tr>
<td>( \vec{V}_4 )</td>
<td>( NPP )</td>
<td>( S_4,S_3,S_5 )</td>
<td>( \vec{V}_4 = \frac{2}{3} * V_g * e^{j3\frac{\pi}{3}} )</td>
</tr>
<tr>
<td>( \vec{V}_5 )</td>
<td>( NNP )</td>
<td>( S_4,S_6,S_5 )</td>
<td>( \vec{V}_5 = \frac{2}{3} * V_g * e^{j4\frac{\pi}{3}} )</td>
</tr>
<tr>
<td>( \vec{V}_6 )</td>
<td>( PNP )</td>
<td>( S_1,S_6,S_5 )</td>
<td>( \vec{V}_6 = \frac{2}{3} * V_g * e^{j5\frac{\pi}{3}} )</td>
</tr>
</tbody>
</table>

The reference vector can be generated by zero-vector and the vector enclosing a sector. The reference vector rotates at an angular velocity of \( 2\frac{\pi}{3} \) times the fundamental frequency of the
inverter output voltage. The inverter output voltage would have rotated one complete cycle if the reference voltage completes one complete revolution thereby creating an AC waveform [11]. Direction of reference vector determines the direction of rotation of motor. [9]

3.4 Calculation of Switching Times:

By rotating the reference vector $\vec{V}_s$ around space vector diagram, modulation can be achieved. By adding all the vectors with in one switching period of $T_s$, modulation can be achieved. Both maximum deviation of current for switching states and cycle time should be small to achieve the required PWM [12]. Vector diagram for calculating duty cycles is as shown in Figure 9.

![Vector Diagram for Calculating Duty Cycles](image)

Figure 9: Generating vector $\vec{V}_s$ from $\vec{V}_1$, $\vec{V}_2$ and $\vec{V}_7$

Duty cycle can be calculated from above. If the vector is in sector 1, the output reference vector can be calculated using (3.7):

$$\int_0^{T_s} \vec{V}_s \, dt = \int_0^{T_1} \vec{V}_1 \, dt + \int_{T_1}^{T_1+T_2} \vec{V}_2 \, dt + \int_{T_1+T_2}^{T_s} \vec{V}_0 \, dt$$  \hspace{1cm} (3.7)

where $T_1$, $T_2$ and $T_0$ are duty cycles of vectors $\vec{V}_1$, $\vec{V}_2$ and $\vec{V}_0$ respectively.
For high switching frequencies, reference vector is constant during time $T_s$; vectors $\vec{V}_1$ and $\vec{V}_2$ are constant and $\vec{V}_0 = 0$ [12]. So, eq (3.7) becomes

$$\vec{V}_s \cdot T_s = \vec{V}_1 \cdot T_1 + \vec{V}_2 \cdot T_2 \tag{3.8}$$

and

$$T_s = T_0 + T_1 + T_2 \tag{3.9}$$

Representing the space vectors in rectangular coordinates, we have

$$\vec{V}_s \ast T_s \ast \left( \begin{array}{c} \cos \theta \\ \sin \theta \end{array} \right) = T_1 \ast \frac{2}{3} V_g \ast \left( \begin{array}{c} 1 \\ 0 \end{array} \right) + T_2 \ast \frac{2}{3} V_g \ast \left( \begin{array}{c} \cos \theta \\ \sin \theta \end{array} \right) \tag{3.10}$$

Solving the above equations:

$$\vec{V}_s \ast T_s \ast \cos \theta = T_1 \ast \frac{2}{3} V_g + T_2 \ast \frac{2}{3} V_g \ast \frac{1}{2}$$

$$\vec{V}_s \ast T_s \ast \sin \theta = T_2 \ast \frac{2}{3} V_g \ast \frac{\sqrt{3}}{2} \tag{3.11}$$

Solving this equation using (3.9) gives

$$T_1 = \frac{\sqrt{3} \ast T_s \ast \vec{V}_s}{V_g} \ast \sin (60 - \theta)$$

$$T_2 = \frac{\sqrt{3} \ast T_s \ast \vec{V}_s}{V_g} \ast \sin (\theta)$$

$$T_0 = T_s - T_1 - T_2 \tag{3.12}$$

With the help of eq (3.12) we can calculate switching times in all the sectors and a multiple of angle $60^0$ has to be subtracted from the actual angle so that the resultant lies in the first sector.
The switching times are calculated using Arduino UNO microcontroller using CORDIC algorithm and these times are used to generate PWM signals which can drive the inverter and make the motor to run.

3.5 Switching Sequence:

Several variations in SVM can be obtained varying null vector from $\mathbf{V}_0$ to $\mathbf{V}_7$ providing different switching performances. Different configurations can be obtained by arranging zero vectors in the seven-segment switching sequence. One of such configuration is shown in the Figure 10.

![Figure 10: Seven-segment switching sequence in sector 1](image)
In the above sequence, all the duty cycles $T_0$, $T_1$ and $T_2$ sum up to get the sampling period $T_s$. The switching frequency will be equal to sampling frequency when the inverter switch turns on and off per sampling frequency. Moving from one sector to other sector doesn’t need any switching [9].

This sequence is used to generate the PWM signals (Table 3) and is fed as input to the motor to make it run. In this thesis switching times are generated using Arduino microcontroller using CORDIC algorithm to compute sine of the angles.

Table 3: Seven Segment Switching Sequence

<table>
<thead>
<tr>
<th>Sector</th>
<th>Switching Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\tilde{v}_0$ $\tilde{v}_1$ $\tilde{v}_2$ $\tilde{v}_0$ $\tilde{v}_2$ $\tilde{v}_1$ $\tilde{v}_0$</td>
</tr>
<tr>
<td></td>
<td>NNN PNN PPN PPP PPN PNN NNN</td>
</tr>
<tr>
<td>2</td>
<td>$\tilde{v}_0$ $\tilde{v}_3$ $\tilde{v}_2$ $\tilde{v}_0$ $\tilde{v}_2$ $\tilde{v}_3$ $\tilde{v}_0$</td>
</tr>
<tr>
<td></td>
<td>NNN NPN PPN PPP PPN NPN NNN</td>
</tr>
<tr>
<td>3</td>
<td>$\tilde{v}_0$ $\tilde{v}_3$ $\tilde{v}_4$ $\tilde{v}_0$ $\tilde{v}_4$ $\tilde{v}_3$ $\tilde{v}_0$</td>
</tr>
<tr>
<td></td>
<td>NNN NPN NPP PPP NPP NPN NNN</td>
</tr>
<tr>
<td>4</td>
<td>$\tilde{v}_0$ $\tilde{v}_5$ $\tilde{v}_4$ $\tilde{v}_0$ $\tilde{v}_4$ $\tilde{v}_5$ $\tilde{v}_0$</td>
</tr>
<tr>
<td></td>
<td>NNN NNP NPP PPP NPP NNP NNN</td>
</tr>
<tr>
<td>5</td>
<td>$\tilde{v}_0$ $\tilde{v}_5$ $\tilde{v}_6$ $\tilde{v}_0$ $\tilde{v}_6$ $\tilde{v}_5$ $\tilde{v}_0$</td>
</tr>
<tr>
<td></td>
<td>NNN NNP PNP PPP PNP NNP NNN</td>
</tr>
<tr>
<td>6</td>
<td>$\tilde{v}_0$ $\tilde{v}_1$ $\tilde{v}_6$ $\tilde{v}_0$ $\tilde{v}_6$ $\tilde{v}_1$ $\tilde{v}_0$</td>
</tr>
<tr>
<td></td>
<td>NNN PNN PNP PPP PNP PNN NNN</td>
</tr>
</tbody>
</table>
CORDIC

CORDIC means COordinate Rotation Digital Computer. It is an iterative algorithm for calculating trigonometric functions, magnitude and phase using simple lookup tables, bit shifts, addition and subtraction operations avoiding multiplications. CORDIC is also used to calculate hyperbolic functions. So it is used for hardware implementations. It was developed by Jack E.Volder in 1959 [13].

CORDIC revolves around the idea of rotation. Compared to other approaches, this is highly recommended when hardware multiplier is unavailable like microcontroller. Table lookup methods and power series are faster than CORDIC when hardware multiplier is available.

4.1 Operation of CORDIC in Rotating Mode:

This algorithm is based on rotation of a vector in a plane (Figure11) [14]. Let \( \vec{R} \) be the vector with initial points as \((X_0, Y_0)\). If the vector is rotated by certain angle \( \theta \), let the new endpoints after rotation be \((X_n, Y_n)\). The algebraic representation of the vector is expressed as:

\[
X_n = X_0 \cdot cos \theta - Y_0 \cdot sin \theta
\]

\[
Y_n = X_0 \cdot sin \theta + Y_0 \cdot cos \theta
\] (4.1)
The magnitude of the vector is assumed to be constant. So, the implementation of CORDIC involves four multiplications and two addition operations and also evaluation of $\cos \theta$ and $\sin \theta$. The multiplications can be eliminated by restricting the angles of rotation to powers of 2 [11].

![Diagram of vector rotation](image)

**Figure 11:** Perfect rotation of a vector in a plane

Angle ‘$\theta$’ can be decomposed into smaller angles and the sum of all angles equals to the original angle:

$$\theta = \sum_{i=0}^{\infty} \theta_i$$

(4.2)

This makes sure that angle ‘$\theta$’ will be equal to the product of all rotations by addition of all the individual angles. By this micro rotation using (4.1), we get

$$X_n = X_0 \cdot \cos \theta_i - Y_0 \cdot \sin \theta_i$$

$$Y_n = X_0 \cdot \sin \theta_i + Y_0 \cdot \cos \theta_i$$

(4.3)
But multiplications need to be eliminated in order to implement CORDIC. So, by modifying (4.3), we get

\[ X_n = \cos \theta_i (X_0 - Y_0 \tan \theta_i) \]

\[ Y_n = \cos \theta_i (X_0 \tan \theta_i + Y_0) \]  

(4.4)

Now the angle \( \theta_i \) is chosen as multiples of \( \tan^{-1}(2^{-i}) \), so that angle \( \tan \theta_i \) becomes \( d_i \times 2^{-i} \) (\( d_i \) takes values +1 or -1). This guarantees that multiplications can be done using shift operations.

Assuming \( \cos \theta_i \) to be constant, which is the scaling factor \( K_i \):

\[ X_n = K_i [(X_0 - Y_0 \times d_i \times 2^{-i})] \]

\[ Y_n = K_i [(X_0 \times d_i \times 2^{-i} + Y_0)] \]  

(4.5)

The original angle is broken down to microangles and hence the relation:

\[ \text{Rotation (}\theta\text{)} = \prod_{i=0}^{\infty} \text{Rotation (}\theta_i\text{)} \]  

(4.6)

\( K_i \) is calculated in advance as single constant if number of iterations is fixed.

\[ K = \prod_{i=0}^{\infty} K_i \]  

(4.7)

But \( K_i = \cos \theta_i \), substituting in the above equation:

\[ K = \prod_{i=0}^{\infty} \cos (\theta_i) \]

\[ K = \prod_{i=0}^{\infty} \cos (\tan^{-1}(2^{-i})) \]  

(4.8)

\[ K = 0.60725293 \] ..........
Even though the angular rotation is done with positive or negative sign, $K$ value remains constant.

Third variable is used to keep track of total rotation to store angle $\theta$ after the decomposition:

$$Z_n = Z_0 - d_i \theta_i$$

$$Z_n = Z_0 - d_i \tan^{-1}(2^{-i}) \quad (4.9)$$

The values of $\tan^{-1}(2^{-i})$ are stored in the lookup table. The complete set of three equations is given as follows:

$$X_n = K_i [(X_0 - Y_0 * d_i * 2^{-i})]$$

$$Y_n = K_i [(X_0 * d_i * 2^{-i} + Y_0)]$$

$$Z_n = Z_0 - d_i \tan^{-1}(2^{-i}) \quad (4.10)$$

where $d_i = \pm 1$.

Equation (4.10) requires addition operation, bit shift operations and look up table. The general implementation of CORDIC is shown in Figure 12. The main aim is to eliminate $Z_n$ and the direction of rotation determines the sign of $d_i$. At iteration ‘i’ (0 < i < n-1) the coordinates become $X_{i+1}$, $Y_{i+1}$ and $Z_{i+1}$, which are given as
\[ X_{i+1} = K_i \left[ (X_i - Y_i \times d_i \times 2^{i}) \right] \]

\[ Y_{i+1} = K_i \left[ (X_i \times d_i \times 2^{i} + Y_i) \right] \]

\[ Z_{i+1} = Z_i - d_i \times \tan^{-1}(2^{i}) \]

(4.11)

Figure 12: General Implementation of CORDIC [11]

4.2 Implementing CORDIC:

Initially vector is rotated by an angle ‘θ’. At each iteration, the direction of rotation is chosen to minimize \( Z_i \) to zero. The equations are given as

\[ X_{i+1} = K_i \left[ (X_i - Y_i \times d_i \times 2^{i}) \right] \]

\[ Y_{i+1} = K_i \left[ (X_i \times d_i \times 2^{i} + Y_i) \right] \]

\[ Z_{i+1} = Z_i - d_i \times \tan^{-1}(2^{i}) \]
The desired angle of rotation is obtained by performing a series of successively smaller elementary rotations, where \( i = 0, 1, 2 \ldots n-1 \).

Let the desired angle be \( 20^0 \), that is, \( Z_i = 20^0 \). For each iteration if \( Z_i > 0^0 \) subtract the iteration angle from \( Z_i \) or else if \( Z_i < 0^0 \) add the current iteration angle to \( Z_i \) and make appropriate \( X_i \) and \( Y_i \) calculations. See Table 4.

Table 4: Lookup Table for the values of \( \tan^{-1}(2^i) \)

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \tan \theta = 2^i )</th>
<th>( \theta = \tan^{-1}(2^i) )</th>
<th>( Z_i )</th>
<th>Rotation</th>
<th>Final Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>45</td>
<td>20</td>
<td>-45</td>
<td>25</td>
</tr>
<tr>
<td>1</td>
<td>1/2</td>
<td>26.565</td>
<td>-25</td>
<td>26.565</td>
<td>1.565</td>
</tr>
<tr>
<td>2</td>
<td>1/4</td>
<td>14.036</td>
<td>1.565</td>
<td>-14.036</td>
<td>-12.47</td>
</tr>
<tr>
<td>3</td>
<td>1/8</td>
<td>7.125</td>
<td>-12.47</td>
<td>7.125</td>
<td>-5.436</td>
</tr>
<tr>
<td>4</td>
<td>1/16</td>
<td>3.576</td>
<td>-5.436</td>
<td>3.576</td>
<td>-1.77</td>
</tr>
<tr>
<td>5</td>
<td>1/32</td>
<td>1.79</td>
<td>-1.77</td>
<td>1.79</td>
<td>0.20</td>
</tr>
<tr>
<td>6</td>
<td>1/64</td>
<td>0.895</td>
<td>0.20</td>
<td>-0.895</td>
<td>-0.875</td>
</tr>
</tbody>
</table>

Table 4 Continued.
With the help of the above implementation, it took almost 20 iterations to calculate angle of $20^0$. Irrespective of direction of rotation, $\cos \theta_i (\cos \theta_i = \cos (-\theta_i))$ becomes constant. So as $Z_i$ approaches zero, the final equations become:

\[
X_n = K \left[ X(0) \cos \theta - Y(0) \sin \theta \right]
\]

\[
Y_n = K \left[ Y(0) \sin \theta + X(0) \cos \theta \right]
\]

\[
Z_n = 0
\]  

(4.12)

$X(0)$ and $Y(0)$ are chosen as $I/K$ and 0 respectively to eliminate the $K$ value. The final values are:

\[
X_n = \cos \theta
\]

\[
Y_n = \sin \theta
\]

(4.13)

Using the Arduino we calculated the angles instead of using long lookup tables which consume a lot of time and also memory. Usually around 15 iterations would give the accurate
results and is used for most of the applications. This algorithm is used to compute trigonometric functions without the use of multiplications. The algorithm converges as long as the rotation angle is within the bounds [14]:

\[ \theta_{\text{max}} = \sum_{i=0}^{\infty} \tan^{-1}(2^{-i}) = 99.88^0. \]

In this thesis CORDIC is used for computing sine waves for calculating switching pulse width modulated waveforms and duty cycles of the inverter switches.

### 4.3 Steps for Implementing CORDIC:

The CORDIC algorithm is used to calculate the duty cycles of stationary vectors. Steps involved in computing CORDIC are given below:

1. Start
2. Compute lookup tables
3. Start with an angle \( Z_i \) = angle , \( x = i/K \) and \( y = 0 \)
4. If \( Z_i > 0 \) then \( d_i = 1 \) or else \( d_i = -1 \)
5. Compute new value ‘\( x \)’, new ‘\( X \)’
6. Compute the new value ‘\( Y \)’ and ‘\( Z \)’
7. Assign to ‘\( x \)’ the new value of ‘\( X \)’
8. Compute new value of ‘\( z \)’ using lookup table
9. If it the end of iteration, return the ‘\( y \)’ value or else go to step 4.
10. Stop
Figure 13 shows the implementation of CORDIC as flowchart.

Figure 13: Flow chart of implementation of CORDIC
CHAPTER-5

RESULTS

Using Arduino SVM is implemented using CORDIC algorithm. The results of using CORDIC to calculate switching times in SVM are given below.

In the Figure 14, CORDIC was implemented for angle 30°. ‘x’ value gives the cosine value and ‘y’ value gives the sine of angle 30°. To justify the use of the CORDIC algorithm, a comparison of the time used in calculating the sine with the CORDIC algorithm to the standard sine function was done. To get an accurate measure of the calculation time, 27 iterations of the CORDIC algorithm was completed. The time taken to complete one-iteration is 1.92 μs which is less than traditional method of implementing sine wave. The time taken for implementing sine wave for different angles using traditional method was 4.90 μs.
Figure 14: CORDIC generated for angle 30°

Figure 15 shows the pulse width modulated signals for sectors 1-6 which are operated at a frequency of 50Hz.
Figure 15: PWM waveforms for frequency 50 Hz using CORDIC
CHAPTER-6

CONCLUSION

Switching cycles of the inverter are generated and space vector modulation has been implemented to calculate the switching times for different frequencies. The pulse width modulated signals were generated using CORDIC algorithm and then fed as input to the motor. All these calculations were implemented using Arduino microcontroller.

CORDIC algorithm takes less time in computing the sine values compared to any other methods which are used to generate sine values. It helps in reducing size of the lookup table and the results are accurate.

6.1 Future Scope of Work:

The work can be extended to control motor speed using V/Hz method and, before changing the speed of the motor, check its limits. Implementation of space vector modulation using unbalanced condition \((V_{AO}(t) + V_{BO}(t) + V_{CO}(t) \neq 0)\) can also be tried.
REFERENCES


[5] Atmega328 datasheet available [online] at :


APPENDIX

CODE LISTING

CORDIC IMPLEMENTATION:

long cordic_lookup [ ] =
{
    0x20000000L,
    0x12E4051EL,
    0x09FB385BL,
    0x051111D4L,
    0x028B0D43L,
    0x0145D7E1L,
    0x00A2F61EL,
    0x00517C55L,
    0x0028BE53L,
    0x00145F2FL,
    0x00A2F61EL,
    0x00517C55L,
    0x0028BE53L,
    0x00145F2FL,
    0x00A2F61EL,
    0x00517C55L,
    0x0028BE53L,
    0x00145F3L,
    0x000A2F3AL,
    0x000517DL,
    0x00028BEEL,
    0x0000145FL,
    0x0000A30L,
#define ITERS 27

void setup () {
    Serial.begin (57600) ;
    long elapsed = micros () ;
    for (long i = 0 ; i < ITERS ; i++)
        elapsed = micros () - elapsed ;
    Serial.print ("time taken for ");
    Serial.print (ITERS) ;
    Serial.print (" iterations = ");
    Serial.print (elapsed) ;
    Serial.println ("us") ;
    Serial.print (elapsed / ITERS) ;
    Serial.println (" us/iter") ;
    test_cordic (0x20000000L, true) ;
}

void test_cordic (long angle, boolean printres) {
    long xx = 607252935L ;
long yy = 0L;
for (int i = 0; i <= 27; i++)
{
    long zi = cordic_lookup[i];
    long xnew = yy >> i;
    long ynew = -xx >> i;
    if (angle < 0L)
    {
        angle += zi;
        xx += xnew;
        yy += ynew;
    }
    else
    {
        angle -= zi;
        xx -= xnew;
        yy -= ynew;
    }
}
if (!printres)
    return;
Serial.print("angle=");
Serial.print("30");
Serial.print("  end x = 0.");
Serial.print(xx);
Serial.print("  end y = 0.");
Serial.println(yy);
}
void loop()
{ }

CALCULATION OF PULSE WIDTH MODULATED SIGNALS USING CORDIC:

```
long cordic_lookup [ ] =
{
  0x20000000L,
  0x12E4051EL,
  0x09FB385BL,
  0x051111D4L,
  0x028B0D43L,
  0x0145D7E1L,
  0x00A2F61EL,
  0x00517C55L,
  0x0028BE53L,
  0x00145F2FL,
  0x000A2F98L,
  0x000517CCL,
  0x00028BE6L,
  0x000145F3L,
  0x0000A2FAL,
  0x0000517DL,
  0x000028BEL,
  0x0000145FL,
  0x00000A30L,
  0x00000518L,
  0x0000028CL,
  0x00000146L,
  0x000000A3L,
  0x00000051L,
  0x00000029L,
```
int ledPin1 = 9;
int ledPin2 = 10;
int ledPin3 = 3;

int val_a = 0;
int val_b = 0;
int val_c = 0;

void setup()
{
    pinMode(ledPin1, OUTPUT);
    pinMode(ledPin2, OUTPUT);
    pinMode(ledPin3, OUTPUT);
}

void loop()
{
    int dt;
    {
        int t1 = 0;
    }
```c
int t2 = millis();
dt = t2-t1;
t1=t2;
return;
}

const float pi = 3.14;
float k= dt*0.1*pi+k;
int n;
float rad = 60 * pi/180.0;
if (k>rad);
{
    k= k-rad;
n= n+1;
}

float Ts = 255;
float T1;
float T2;
float T0;

T1 = sqrt(3)*Ts*sin(rad*n-k);
T2 = sqrt(3)*Ts*sin(k-(n-1)*rad);
T0 = Ts-T1-T2;
val_a = T1+T2+(T0/2);
val_b = T2+(T0/2);
val_c = T0/2;
```
analogWrite(ledPin1, val_a);
analogWrite(ledPin2, val_b);
analogWrite(ledPin3, val_c);
}

// Calculation of sine values using CORDIC //
void test_cordic (long aa, boolean printres)
{
    long xx = 607252935L;
    long yy = 0L;
    for (int i = 0 ; i <= 27 ; i++)
    {
        long zi = cordic_lookup [i] ;
        long tx = yy >> i ;
        long ty = -xx >> i ;
        if (aa < 0L)
        {
            aa += zi ;
            xx += tx ;
            yy += ty ;
        }
        else
        {
            aa -= zi ;
            xx -= tx ;
            yy -= ty ;
        }
    }

    if (!printres)
        return ;
}
Serial.print ("angle=") ;
Serial.print ("   end y = 0.") ;
Serial.println (yy) ;
}