5-2023

Embedded Solar Tracker Design Utilizing Solar Position Calculation with Sensor Correction

Austin Martinez
adm1045@rit.edu

EMBEDDED SOLAR TRACKER DESIGN UTILIZING SOLAR POSITION CALCULATION WITH SENSOR CORRECTION

by

AUSTIN MARTINEZ

GRADUATE PAPER

Submitted in partial fulfillment
of the requirements for the degree of
MASTER OF SCIENCE
in Electrical Engineering

Approved by:

____

Mr. Mark A. Indovina, Senior Lecturer
Graduate Research Advisor, Department of Electrical and Microelectronic Engineering

____

Dr. Ferat Sahin, Professor
Department Head, Department of Electrical and Microelectronic Engineering

DEPARTMENT OF ELECTRICAL AND MICROELECTRONIC ENGINEERING
KATE GLEASON COLLEGE OF ENGINEERING
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

MAY, 2023

Dedication

I dedicate this work to my mother Julie Martinez, my father Danny Martinez, my brother Aiden Martinez, and my great friends for their love and support during my time in college.

Austin Martinez

Declaration

I hereby declare that except where specific reference is made to the work of others, that all content of this Graduate Paper are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This Graduate Project is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Austin Martinez

May, 2023

Acknowledgements

I would like to thank my advisor Professor Mark A. Indovina for his support, guidance, feedback, and encouragement which helped in the successful completion of my graduate research. I would also like to thank Professor Carlos Barrios for sparking my interest in embedded systems and digital systems design.

Austin Martinez

Abstract

As renewable energy becomes more accessible, affordable optimization solutions have become a larger part of at home systems. Solar energy is one of the most common clean energy applications used by homeowners because of its increasing affordability. While many photovoltaic systems are statically mounted on roofs or in yards, many systems also employ some kind of solar tracking. Both single axis and dual axis solar tracking systems have been implemented and tested with multiple kinds of tracking methods. Two of the most accurate tracking strategies are utilizing light dependent resistors or using some kind of solar position calculation. This paper proposes a design for a solar tracking system that combines these two strategies, using a solar position algorithm to calculate the angle of the sun which is then corrected by light dependent resistors if necessary. This design is highly accurate as well as being robust in finding the sun regardless of weather conditions and geographical location. Data from two days of testing shows an average percent difference in solar energy output of 1.486% in the late evening and 0.883% near solar noon.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Clean energy is an ever growing industry as it becomes both more energy efficient and more cost effective. Solar has quickly garnered a reputation as being one of the most popular choices of clean energy. Used by big businesses and homeowners alike, improvements can always be made to escalate the efficiency of solar panels for all applications. As more individuals acquire their own panels for private use, affordable optimization strategies are going to be needed.

Many homeowners already take advantage of static solar panels–solar panels that are placed at a permanent angle–however some may want to transition to a system that utilizes a solar tracking system. In theory, a solar panel utilizing a tracking system should absorb more solar energy than a static solar panel. This is discussed in more detail in Section 2. For that reason, people may want to make use of an affordable solar tracking system.

While the sun can be tracked numerous different ways, this project will test the reliability of using Global Positioning System (GPS) data to calculate the angle of the sun. This algorithm, discussed in more detail in Section 3, will use latitude, longitude, and time data from an embedded GPS module to calculate the angle of the sun from the current location. This is a more robust method than light detection, for example, as it is unaffected by cloud coverage.

1.1 Research Goals

The main goal of this design is to implement an affordable and efficient solar tracking system that absorbs more energy than static solar panels by combining two very robust solar tracking techniques. This design will be physically assembled using low-cost materials, as well as programmed in C on a STMicroelectronics (STM) development board. The solar tracking system will be compared to a non-tracking system to verify that the former does in fact increase the energy absorbed by the solar panel.

1.2 Organization

The structure of the paper is as follows:

- Section 2: This section discusses the concept of solar tracking and its comparison to static solar panels. It also speaks on previous work done in this area and how their results influenced this design.
- Section 3: This section covers the algorithm used for calculating the angle of the sun for this project.
- Section 4: This section explains the proposed physical architecture of the system as well as the structure of the software that will be used to control the entire system.
- Section 5: This section details the experiment setup and how the data is gathered for characterization.
- Section 6: This section discusses the data acquired from the experimental setup, namely the comparison between the energy absorption of the tracking system and the non-tracking system.

- Section 7: This section concludes the paper and outlines the possible directions for future work in this area.

Chapter 2

Bibliographical Research

This section discusses the value of solar tracking as a function of capturing solar energy as well as prior work in developing affordable Photovoltaics (PV) systems. Of the examples presented, some utilize some form of solar tracking, and some that do not.

2.1 The Concept of Solar Tracking

The goal of any clean energy system should be to generate as much energy as possible from the renewable source it is targeting. With solar energy, the most absorption occurs when the angle of incidence of the sun's rays is perpendicular to the solar panel [9]. Due to the way the sun passes through the sky as the day advances, the aforementioned angle of incidence is only perpendicular for a portion of the day. Because of this fact, PV systems that rely on a static solar panel are not absorbing all of the energy that could be captured throughout the course of the day. This situation has given way for the development of solar tracking systems that attempt to absorb as much energy as possible.

Solar tracking systems typically come in the following forms:

- Single axis tracker - Horizontal
- Single axis tracker - Vertical
- Dual axis tracker

Examples of these can be found in Figure 2.1.

Single axis trackers, coming in two main forms, try to angle the solar panel toward the sun while only being able to rotate on one axis. In the case of horizontal single axis trackers, the axis of rotation is parallel to the ground, typically rotating the solar panel from east to west as it follows the sun. In the case of vertical single axis trackers, the axis of rotation is perpendicular to the ground, spinning the solar panel like a top. This configuration usually involves a tilted solar panel as well. When compared to static solar panels, single axis tracking systems can perform upwards of 20% better in terms of daily output power [10]. While this higher performance and efficiency may have a higher upfront cost than a static solar panel, the higher energy output will add up over years and eventually save the user money [9]. For that reason, single axis trackers are a great option for a more optimized home PV system.

Dual axis trackers are an expansion on single axis trackers, wherein the two main configurations of single axis trackers are combined into one system. Therefore, the typical dual axis tracker has both a horizontal and vertical axis to rotate about, enabling more precise tracking. Where single axis trackers can be angled closer to the sun than a static solar tracker, they still do not maximize the absorption at all times of the day. Where single axis trackers are a good solution, dual axis trackers prove to be a more optimized solution. This claim is discussed further in Section 2.2 where prior work on solar trackers is covered, speaking on both single and dual axis tracking systems.

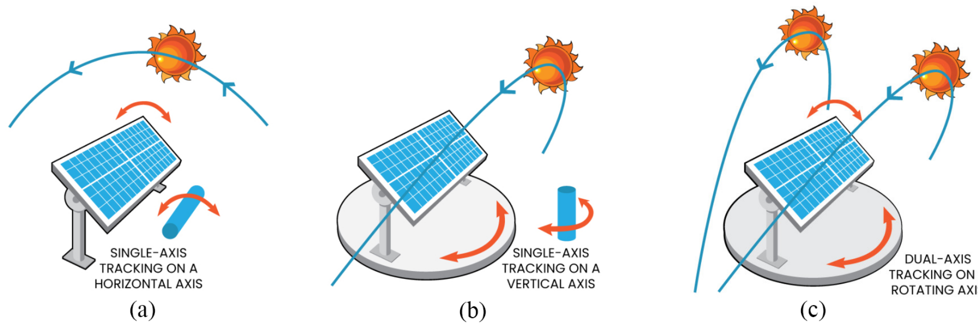

Figure 2.1: Three different types of solar trackers: (a) Single axis horizontal, (b) Single axis vertical, and (c) Dual axis [1, 2].

2.2 Prior Work

The problem of solar tracking has been approached before from many different angles. The prevailing strategies are typically one of the following or a combination of multiple:

- Light detection with light dependent resistor(s) (LDR)
- Angle calculation using Solar Position Algorithm (SPA)
- Maximum power point (MPP) detection
- Incrementally moving along a fixed path given known time and location data

All of these strategies have merit as solutions to solar tracking, however each comes with its own set of advantages and disadvantages.

Many designs implement one or more LDRs as part of the solar tracking system. For some, an LDR is used simply to detect whether the sun has risen or if it is night time. This implementation essentially wakes up the system so that the tracker can take over. For most others, however, the LDRs are used as a means for tracking the sun in the sky. This kind of tracking is usually done using two sensors in the case of single axis trackers, or four sensors in the case of dual axis trackers. The tracking is typically accomplished by placing the sensors in an array and querying them on a regular basis. The resulting values are then used to determine which direction detected the greater light intensity. This determination is then used to move the solar panel in that direction until the values on the sensors are equal, indicating that the panel is pointing directly toward the sun. In single axis tracker experiments, the two LDRs are used to track the sun's movement from east to west throughout the day [9, 11]. With this tracking configuration, the north-south position can be adjusted manually as in [9], or not at all, depending on the desired results. If this angle is not changed by an outsider monthly, it may not be able to achieve maximum solar energy absorption as the sun's path through the sky changes. Nevertheless, these two single axis experiments saw a difference of about 1V of open circuit voltage, with the tracking system having the higher voltage output over the static setup. In a similar experiment that used a simulation that essentially mimicked the single axis two LDR system, similar improvements were seen [12]. In several cases of dual axis trackers using LDRs, the performance observed is even higher. For example, in one experiment that achieved solar tracking using only these sensors, both a single axis and dual axis setup were tested in parallel [10]. These authors found that the increased output power gain was increased by 23% using a single axis tracker, and 40% using a dual axis tracker. Even after accounting for the power used to move the motors, the increased power yield was high. Similar experiments [13, 14] employed the same kind of sensor array technique and observed equally promising results, demonstrating the reliability of LDR sensors in solar tracking. Additionally, the authors of [14] make a point that despite the system's best performance being on clearer days as opposed to cloudy, the performance over the course of a month still outdoes that of a static solar panel. Another related experiment employs the use of LDR sensing coupled with the MPP technique [15]. This setup utilized three sensors as well as four small solar panels as sensors in addition to the main solar panel. The four smaller solar panels are used as the four LDR sensors have been in the aforementioned experiments, using power generation measurements rather than light intensity measurements. However, once the system reached the position calculated by the solar panels, the LDRs were used as a checking mechanism to ensure that the right position was found. If that was not the case, these sensors were used to correct the position. In this paper, this is the introduction of the use of LDRs as a means of correction. The MPP technique is also employed in [16] with the use of a single solar panel. The single axis system performs a periodic "current sweep" wherein the solar panel is swept across its range until the MPP is detected. The system then fixes the solar panel in this position until the next sweep. In other experiments [17, 18], authors paired the LDRs with a proportional-integral-derivative (PID) controller for more sophisticated sensor tracking. These two designs had great success in using these PID systems to track the sun accurately.

Another common technique employs the calculation of the solar position using the time and geographical location of the system. This strategy is used exclusively in [19, 20], and closely modeled in [21]. In [19, 20], single axis and dual axis respectively, a real-time clock (RTC) is used with the geographical location of the system hardcoded into the program to calculate the solar position. Both of these methods achieved success in solar tracking, however [19] made a point that the accuracy of the system is reliant on the accuracy of the mechanical setup. This implies that if at some point there occurs an error in the physical hardware, e.g. the motor housing, the accuracy of the RTC system declines. Because there are no other sensors on board to check the accuracy of the calculated position, the chance of faulty tracking becomes nonzero. Additionally, with both of these programs needing the geographical coordinates to be provided by an outsider or hardcoded, the system becomes less universally applicable. The same principle is used in [22, 23], where the geographical coordinates are provided to the system. However, these two implementations use LDRs as a corrective system for the solar position calculated by the SPA. This is similar to the principle introduced in [15] where the LDRs are used for corrective measures. Yet still, the systems require the coordinates to be provided. While this is not inherently a bad result, it is more robust for the system to be able to determine the geographic location on its own. This leads to the GPS method presented in [24] that uses a GPS receiver to determine the time and geographic location. This information is then sent into the SPA which yields results that can point the solar panel toward the sun. The GPS system is then entirely self-sufficient, able to be moved to a different location and still calculate the correct solar position. This paper also presents a strictly timed method of solar tracking. In this mode, the system waits for the LDR to detect the beginning of the day, at which point the timer system begins moving the motor one degree every hour until the LDR detects that it is night time. This timed method is also implemented in [25] where the system moves every 30 minutes. While it does track the sun in these circumstances, this timer method is very application specific and not as universally applicable as the other methods.

Chapter 3

Solar Position Algorithm

A robust algorithm is required to reliably track the position of the sun in the sky. The National Renewable Energy Laboratory (NREL) has published the Solar Position Algorithm (SPA) that is used for many different applications today. This algorithm, valid for any time from the year -2000 to 6000 can be used to determine the solar zenith and azimuth angles with uncertainties of $\pm 0.0003°$[26]. With that, this is by far one of the most accurate algorithms for such a calculation, and an even more important tool for a solar tracking PV system. The following are the steps that are to be implemented in the proposed solar tracker design.

3.1 SPA Presented by NREL

The Solar Position Algorithm provided by NREL follows a very specific procedure to determine the relevant time and location data necessary to calculate the position of the sun. These steps are listed below and outlined in greater detail in [26].

- Calculate the Julian Day
- Calculate the Julian Ephemeris Day

- Calculate the Julian Century and the Julian Ephemeris Century
- Calculate the Julian Ephemeris Millennium
- Calculate the Earth heliocentric longitude, latitude, and radius vector
- Calculate the geocentric longitude and latitude
- Calculate the nutation in longitude and obliquity
- Calculate the true obliquity of the ecliptic
- Calculate the aberration correction
- Calculate the apparent sun longitude
- Calculate the apparent sidereal time at Greenwich at any given time
- Calculate the geocentric sun right ascension
- Calculate the geocentric sun declination
- Calculate the observer local hour angle
- Calculate the topocentric sun right ascension
- Calculate the topocentric local hour angle
- Calculate the topocentric zenith angle
- Calculate the topocentric azimuth angle
- Calculate the incidence angle for a surface oriented in any direction

3.2 SPA Programs

Algorithms for calculating the solar position have been implemented in programs before. A related solar position algorithm, The Grena's fifth algorithm, is implemented in Python 3.5 to achieve similar accuracy as the NREL algorithm in [8]. The drawback of this algorithm is that the validity only stretches from 2010 to 2110. Nonetheless, this implementation extracted input data from a GPS module. The only data needed from a GPS module to run the SPA is the UTC time and the geographical coordinates. This is demonstrated in the paper. The Python program uses the GPS data and runs their SPA calculation. The program achieved accurate results, calculating the solar angle within less than 0.5° and the solar azimuth angle within less than 0.1° [8]. This type of accuracy is more than acceptable for PV systems which brings confidence to the proposed design.

Another similar algorithm for determining the position of the sun is reviewed in [27]. This paper discusses several of the important parts of the calculation, what data is required, and what results from each of these parts. This is helpful in understanding the different parts of the SPA presented by NREL.

Chapter 4

System Architecture

The proposed design makes use of several of the strategies mentioned in Section 2.2 in an effort to make use of the techniques that have previously yielded positive results. This includes the utilization of the Solar Position Algorithm to calculate the azimuth and elevation angles for the motors to direct the solar panel towards. The information required for this calculation will be acquired through the use of a GPS device to ensure the system is more robust, as mentioned in Section 2.2. Additionally, LDRs are used to verify the solar position, correct the calculated position if necessary, as well as to detect if the sun has risen indicating that the system can wake up.

The design is comprised of both physical hardware and embedded software. The details of both of these parts are discussed in the next two subsections, first with a breakdown of the physical components that make up the hardware setup followed by a detailed explanation of the control program.

4.1 Hardware

The following are the different hardware components that have been used in the proposed design.

4.1.1 STM32 Nucleo-64 Development Board

This development board provides a well put together embedded platform to interface with all of the different components needed for this design. The board has an STM32L476RG microcontroller (MCU) with several useful peripherals such as timers, serial UART ports, ADCs, and DACs. This board will be powered using the USB port for the scope of this project, however in the future it is possible that this could run on energy generated by the solar panel.

4.1.2 MTK3339 GPS Module

The GPS module used for the proposed design is a small low-power chip that has a built-in antenna. The chip has been provided on a breakout board for ease of use and integration into the design. This breakout board also allows the ability to connect a coin cell battery to make use of the real-time clock (RTC) for warm starts. The chip prints out the GPS messages in serial UART using the NMEA 0183 sentence structure. This is discussed further in Section 4.2. For the purposes of this design, the only pins used are the VIN, GND, and TX. The TX pin, in this case, is the pin that transmits the output data from the GPS module to the Nucleo board.

4.1.3 MG996R Servo Motors

Two servo motors are necessary for the proposed design's dual axis tracking system. These motors are controlled by pulse width modulation (PWM). By sending a certain frequency to the motor, it will turn to a certain angle. This control will allow for precise tracking of the sun. These two motors are held together in a 3D printed dual axis housing that allows for one of them to control the horizontal rotation and the other to control the vertical rotation. The solar panel is then mounted on the outer edge of the vertical motor.

4.1.4 Light Dependent Resistors (LDR)

The proposed design uses four LDRs for verifying and correcting the solar position calculated by the SPA. The LDRs used in the proposed design have a resistance range of 1KΩ to 10KΩ. The resistance of these components change depending on the amount of light that the surface is exposed to. The four LDRs are placed in a diamond-like grid pointing in the same direction as the solar panel and are all separated by an opaque wall. This achieves independent readings by each sensor such that if they are facing at an angle away from the sun, some of them are in the light and some of them are in the dark. This produces a measurable difference in their respective resistance values which then allows for the determination of the true direction of the sun. If the sensors all have marginally similar readings, however, this verifies that the solar panel is pointed in the direction of the sun.

4.1.5 Solar Panel

The solar panel used for the proposed design is a smaller solar panel, capable of 100 mA at 5 V. This small solar panel is chosen to have a smaller load on the servo motors as well as the ADC on the Nucleo board.

4.2 Software

The flow of the control program is shown in Figure 4.1. The relevant source files can be seen in Appendix B. The control program consists of three main operations: wake up detection, solar tracking, and sunset detection.

4.2.1 Wake Up Detection

This part of the program is responsible for determining when the sun comes up and the day has started. This is done by regularly querying the LDRs to see if the measured light value suggests that the sun has risen. If the sun is not yet detected, the system goes back into deep sleep for another 30 minutes, when the sensors will be queried once again.

4.2.2 Solar Tracking

This component makes up the majority of the control program. Once the system has woken up, the GPS module will be queried for the first time using the GPS_receive function. The function works by reading the serial UART output from the GPS module until a full NMEA sentence is detected. The function then checks if the sentence received was of the GA type or RMC type. If it is not, the function will continue to read the serial UART output and check until the GA sentence is detected. If the GA sentence is detected, the parseGGA function is called which will decode the sentence and extract all of the data that it sent, as shown in Table 4.1. The same occurs for the case of detecting the RMC sentence, the breakdown of which can be seen in Table 4.2. Once the GPS data has been parsed, the function is exited and the data is passed to the SPA calculation. This function takes in the UTC time, latitude, and longitude, and returns a zero if the calculation was done without errors. From this, the zenith angle and azimuth angle are used to calculate the PWM values necessary to angle the two servo motors correctly. These values are then sent to the corresponding timer channels which effectively moves the motors to the desired locations. From there, the LDRs are queried to determine if the calculated angle is correct. This is done by checking that all four LDRs have values that are marginally similar to each other. If the differences between the values are within a predetermined threshold, the function returns. If the difference between two values is greater than a predetermined threshold, then their values are used to correct the position. This correction is done as shown in Figure [LDR code snippet]. The motors are moved in the direction of greater light values until the LDR readings are within the difference threshold once again. Once the solar panel positioning is finished being determined, the system enters a wait period. During this wait period, the system collects solar energy for five minutes. At the end of this period, the system runs the sunset detection, as shown in Figure 4.1.

4.2.3 Sunset Detection

This part of the program is done at the end of every loop of the main solar tracking functionality. After each wait period, the LDR sensors are queried to get light values. These values are then checked to see whether they are less than a valid value, indicating that the sun is no longer high enough in the sky to be used. If this is true, the system resets the motor positions, enters a deep sleep, and goes back into the wake up detection functionality. Otherwise, the system goes back to the top of the solar tracking as shown in Figure 4.1.

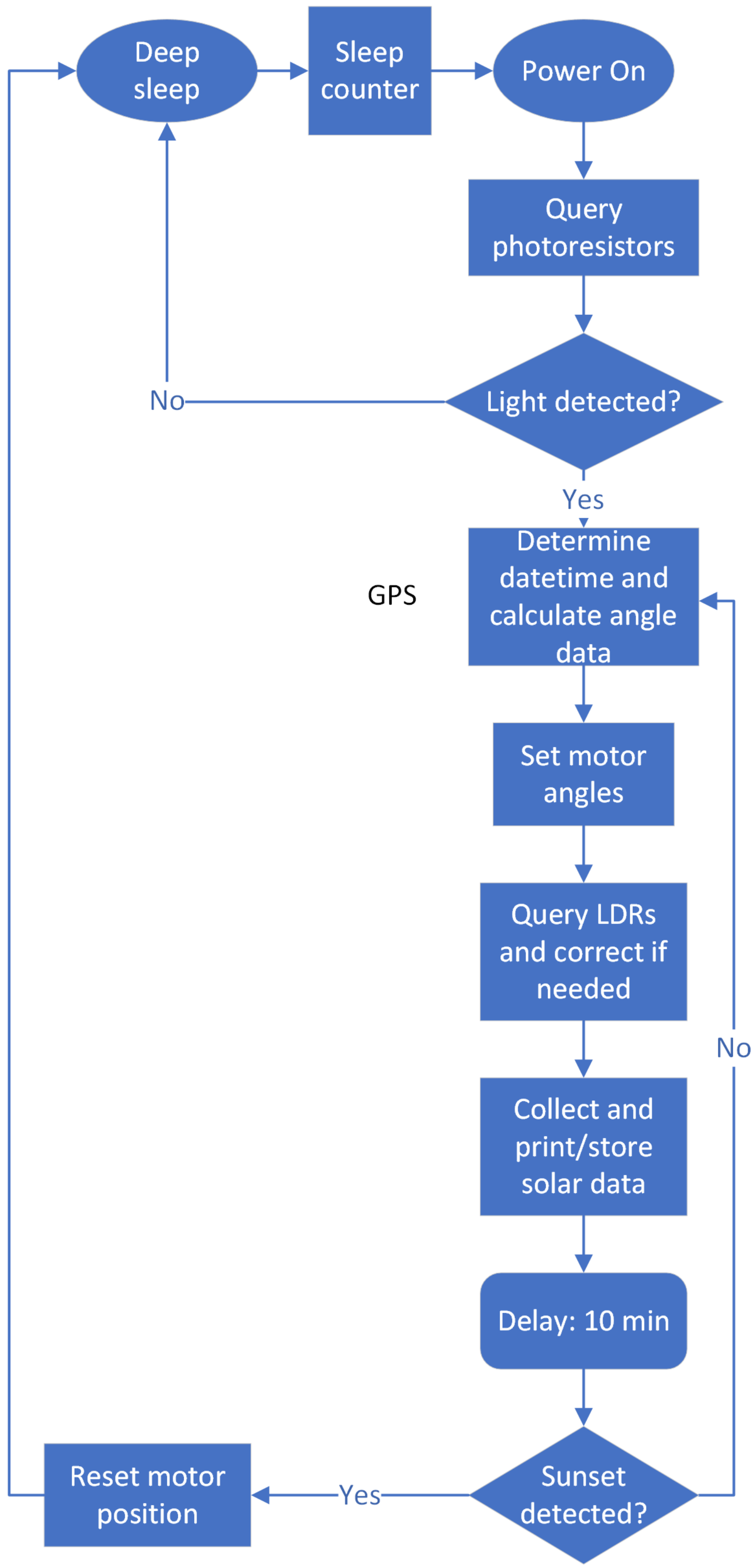

Figure 4.1: Program overview flowchart

Table 4.1: GPS GGA NMEA sentence structure [8]

Data	Structure
UTC time of position fix	hhmmss.ss
Latitude	ddmm.mmmm
Hemisphereical Orientation N/S	n
Longitude	dddmm.mmmm
Hemispherical Orientation W/E	w
Type of position fix	x
Number of satellites used	y
Horizontal dilution of precision	z.z
Sea level altitude	aaaa.a
Altitude units	m
Geoidal separation	g.g
Geoidal separation units	m

Table 4.2: GPS RMC NMEA sentence structure [8]

Data	Structure
UTC time of position fix	hhmmss.ss
Status	a
Latitude	ddmm.mmmm
Hemisphereical Orientation N/S	n
Longitude	ddmm.mmmm
Hemispherical Orientation W/E	w
Speed over ground (knots)	z.z
Track made good (degrees)	y.y
UTC date of position fix	ddmmyy
Magnetic Variation (degrees)	d.d
Variation Sense	v

Chapter 5

Experimental Setup

This section discusses the physical experimental setup, the type of data collected, and how the data is collected.

5.1 Physical Setup

The proposed design is assembled as shown in Figure 5.1. The ST Nucleo-64 development board receives 5 V power from a laptop. The board distributes 5 V power to the two MG996R servo motors and 3.3 V power to the GPS device and the LDRs. The UART TX port of the GPS device is connected to the UART1 RX port of the development board to receive the NMEA sentences containing the GPS data. The four LDRs are being pulled up to 3.3 V using 1KΩ resistors. The LDR values are being probed on the net between the LDR and the pull-up resistor for each, labeled as LDRn where $n \in \{1,2,3,4\}$. These probes are wired into four channels of ADC1 on the development board. The ADC converts the voltage into a digital value that is used in the program. The servo motors' PWM lines are connected to Timer 3 Channel 1 and Channel 2 ports on the development board. This is how the motors receive

PWM signals that control their positions. The two servo motors are placed in a dual axis housing that allows them to rotate to any solar angle necessary. The fixture that holds the solar panel and the LDR circuit is attached to the vertical motor as that is the one that will point the fixture toward the sun. Additionally, a voltage divider is used to scale the output voltage of the solar panel from 5.5 V to 3.5 V as the reference voltage for the development board is 3.5 V. This scaled output voltage is connected to another input channel of ADC1 on the development board. The schematic for this physical setup is shown in Figure 5.2

5.2 Data Collection

The relevant data collected from the experimental setup includes the output voltage of the solar panel and the accuracy of the GPS calculated positioning. Two key data points are collected for the output voltage at each newly calculated position. The first data point is the output voltage of the solar panel in the position calculated by the proposed design. The second data point is the output voltage of the solar panel pointed straight up into the sky, simulating a static solar panel. The accuracy of the GPS calculated positioning is also tracked each time the system moves the panel to a newly determined position. This is a binary data point detecting whether the calculated position is pointing the solar panel directly at the sun or not. Finally, the date and time is collected at each new position to understand how the data changes throughout the day.

5.3 Procedure

The system is tested using the physical setup described above. The solar tracking system is placed in one spot for a duration of time, allowing the system to run and continuously collect data. All of the data is printed to a terminal using UART and continually logged in a text file.

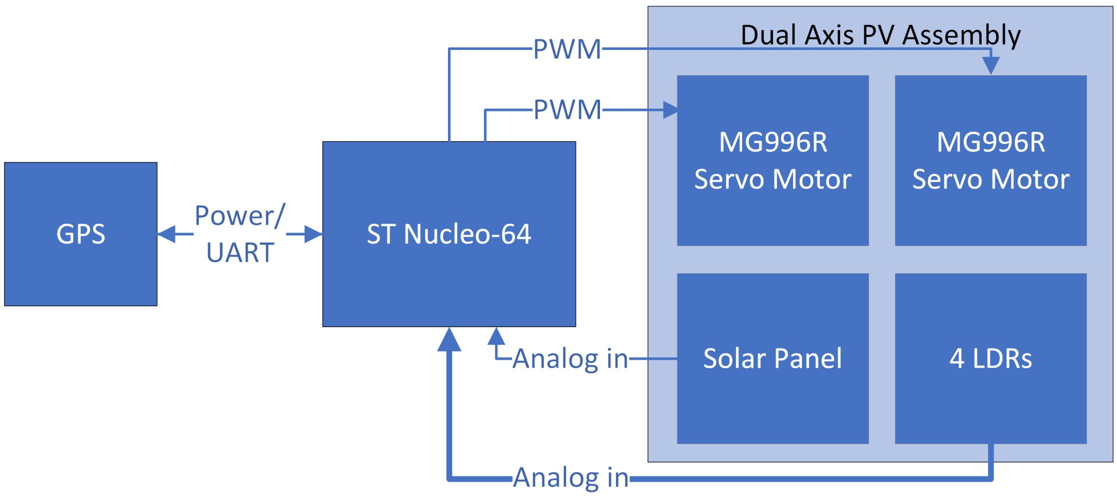

Figure 5.1: Experimental setup of proposed design.

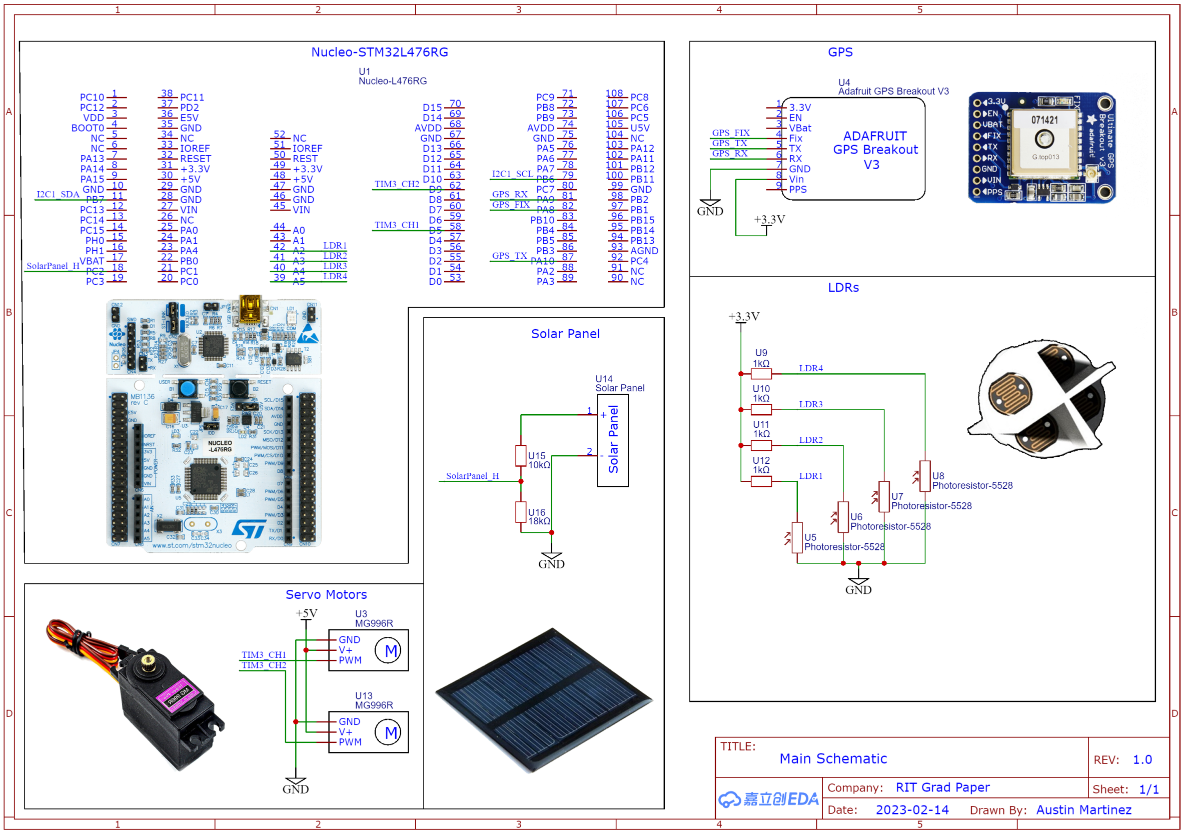

Figure 5.2: Schematic of the physical setup, describing the connections between each device [3–7].

Chapter 6

Results and Discussion

The results of the experiment and analyses of the results is provided in this chapter.

6.1 Solar Data

The experimental setup was run for a total of 116 minutes over two days where the weather was partly sunny. The solar data collected from these two days is shown in Figures 6.1 and 6.2. The solar data collected on May 1 covers the times between 5:23 PM and 6:56 PM Eastern Time (ET). As shown in the graph, the sun is declining at this time of the day. This is evident from the decreasing values collected for the Output Voltage as the time progresses. The solar data collected on May 2 covers the times between 1:01 PM and 1:16 PM ET. As shown in this graph, the sun is almost directly above the experimental setup as the solar data does not change very much in the time interval. This is confirmed as the SPA calculated the solar noon–the time when the sun reaches its highest point in the sky–to be at 1:08 PM on May 2. In both figures, the solar energy being absorbed in the position calculated by the SPA is represented by the blue line, whereas the solar energy being absorbed by the simulated static solar panel is represented by the red line. Both figures show that the output voltage of both positions is quite similar for most of the duration of the test. Table 6.1 shows that the average percent difference of the output voltage of the two solar panel positions is less than two percent for both days' data.

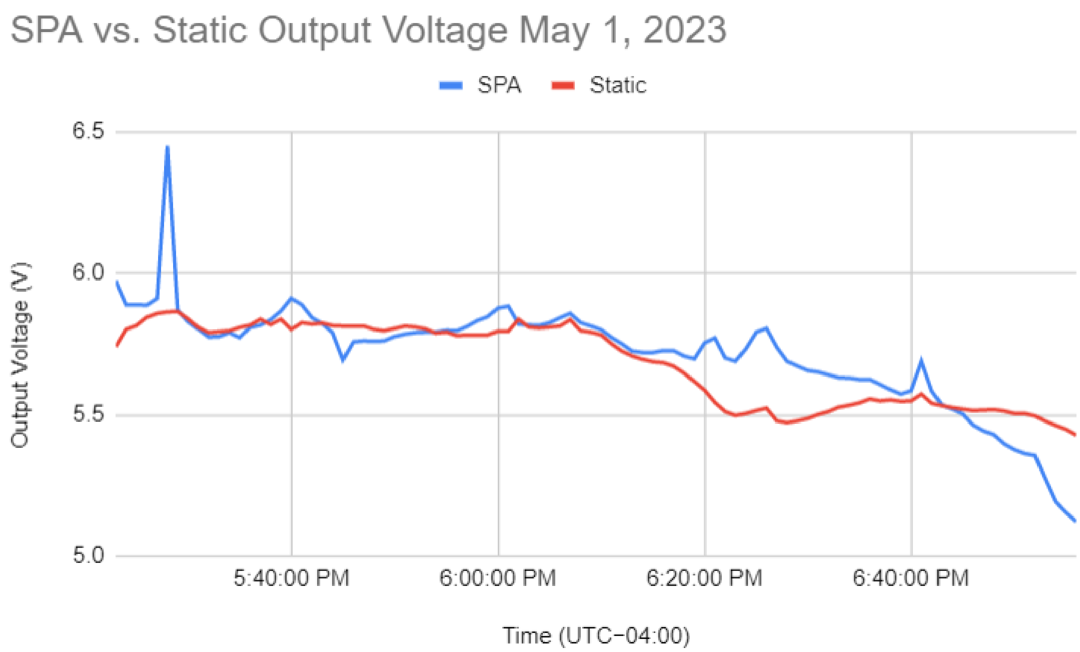

Figure 6.1: Solar data collected on May 1, 2023.

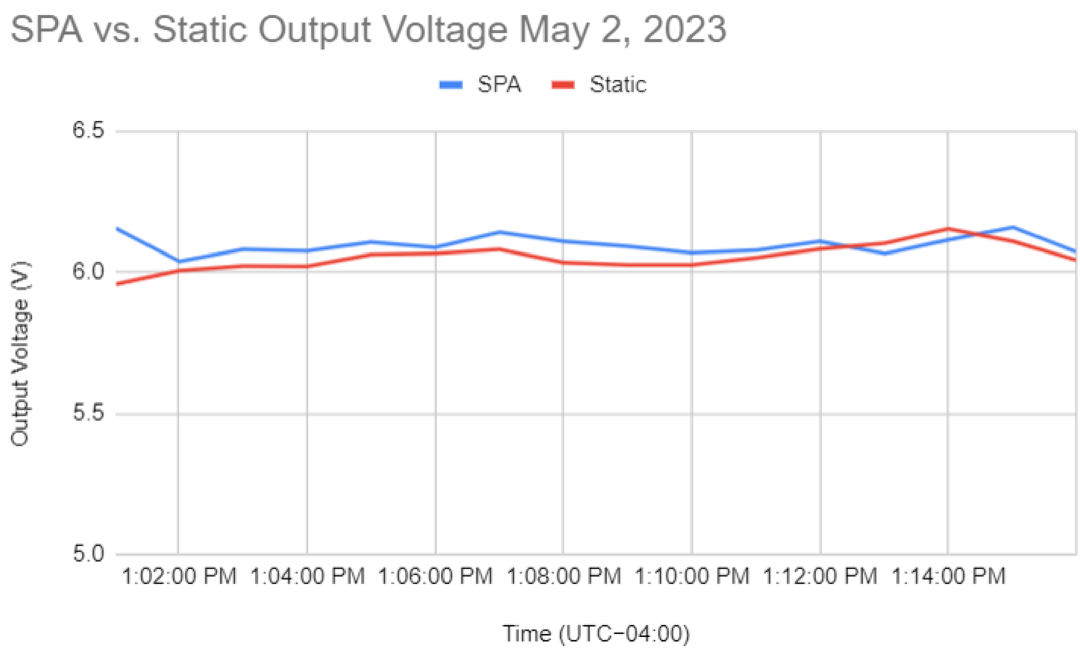

Figure 6.2: Solar data collected on May 2, 2023.

Table 6.1: Average Percent Difference SPA vs. Static Output Voltage

Day	Average Percent Difference
May 1, 2023	1.486%
May 2, 2023	0.883%

Chapter 7

Conclusion

This paper proposes a design for a Photovoltaics system that uses GPS data coupled with a Solar Position Algorithm to increase the energy output. The system accomplishes this by accurately tracking the position of the sun and aiming the solar panel in that direction, maximizing the possible solar absorption. This position is then checked using light dependent resistors to ensure that the calculated position is true. Solar tracker systems have been proposed in many different forms, as was described earlier in this paper, however this design combines the accuracy of precise calculations with the reliability of self-checking. The data shows that the solar tracking is accurate and that the energy output is improved.

The goal of the research was to create an affordable and easy-to-use system that would accurately track the sun. The proposed design is programmed in C on an affordable development board. The entire system, with the exception of the solar panel, can be assembled for just over \$100. Other than assembly, the most a user would need to change would be the UTC offset.

7.1 Future Work

While the work described in this paper was largely successful, there are multiple areas where the proposed design can be improved. A few suggestions for possible improvements are as follows:

- The program is written specifically for the STM32 Nucleo-L476RG, which has a pin layout and peripheral setup that may differ from other STM development boards. The program can be expanded to support other development boards.
- While the system was proven to outperform a static solar panel, the size of the panel used in the experimental setup may have limited the results. A future experimental setup could utilize a larger solar panel that may yield better results.
- The current program is only configurable by editing the code. The design could be expanded to be configured in some sort of user interface, either through a serial terminal, screen, or otherwise.
- The GPS device used in the proposed design is accurate and provides data within 30 seconds or after 5 minutes, depending on the weather conditions. A future iteration of the design could use an external antenna to get GPS data faster if weather conditions prove to be a problem.
- The experimental setup was not tested in harsh conditions like rain or snow. This could be useful data to acquire.

References

[1] D. De Rooij, “Single Axis Trackers,” sinovoltaics.com. [Online]. Available: https://sinovoltaics.com/learning-center/csp/single-axis-trackers/

[2] Niclas, “Dual Axis Trackers,” sinovoltaics.com. [Online]. Available: https://sinovoltaics.com/learning-center/csp/dual-axis-trackers/

[3] botland, “STM32 NUCLEO-L476RG - with STM32L476RGT6 ARM Botland - Robotic Shop,” botland.store. [Online]. Available: https://botland.store/stm32-nucleo/18800-stm32-nucleo-l476rg-with-stm32l476rgt6-arm-cortex-m4-mcu-5904422364977.html

[4] Deltakit, “6V Mini Solar Panel 6V 80mA Mini Solar Panel DIY (7cm by 7cm),” Deltakit. [Online]. Available: https://www.deltakit.net/product/6v-80ma-mini-solar-panel-diy-7cmx7cm/

[5] Electropeak, “TowerPro MG996R 55G High Torque Digital Servo,” electropeak.com. [Online]. Available: https://electropeak.com/towerpro-mg996r-55g-high-torque-digital-servo

[6] L. Fried, “Adafruit Learning System,” learn.adafruit.com. [Online]. Available: https://learn.adafruit.com/assets/31842

[7] "Solar Tracking - FennecLabs," shopz.off-75.ml. [Online]. Available: https://shopz.off-75.ml/ProductDetail.aspx?iid=430990134&pr=63.88

[8] J. Fonseca-Campos, L. Fonseca-Ruiz, and P. N. Cortez-Herrera, "Portable system for the calculation of the sun position based on a laptop, a GPS and Python," in *2016 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC)*, 2016, pp. 1–5.

[9] Reza, Md. Nasim and Hossain, Md. Sanwar and Mondol, Nibir and Kabir, Md. Alamgir, "Design and Implementation of an Automatic Single Axis Solar Tracking System to Enhance the Performance of a Solar Photovoltaic Panel," in *2021 International Conference on Science and Contemporary Technologies (ICSCT)*, 2021, pp. 1–6.

[10] W. A. Shah, M. W. Khan, R. Mansoor, and Arshad, "Analysis of Power-Efficient High Torque Solar Tracker through PID Controller," in *2020 7th International Conference on Electrical and Electronics Engineering (ICEEE)*, 2020, pp. 176–180.

[11] A. Kulkarni, T. Kshirsagar, A. Laturia, and P. H. Ghare, "An Intelligent Solar Tracker for Photovoltaic Panels," in *2013 Texas Instruments India Educators' Conference*, 2013, pp. 390–393.

[12] C. Alexandru and C. Pozna, "Different tracking strategies for optimizing the energetic efficiency of a photovoltaic system," in *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, vol. 3, 2008, pp. 434–439.

[13] S. Siddula, C. Dennis Gleeson, and P. Geetha kumari, "Solar Panel Position Control and Monitoring System For Maximum Power Generation," in *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, 2020, pp. 169–174.

[14] A. Masih and I. Odinaev, “Performance Comparison of Dual Axis Solar Tracker with Static Solar System in Ural Region of Russia,” in *2019 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, 2019, pp. 375–378.

[15] S. Kumar, A. Kumar Pal, P. Singh, S. Mittal, and Y. Kumar, “Solar probe based autonomous solar tracker system - A review,” in *2021 International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, 2021, pp. 589–594.

[16] F. Huang, D. Tien, and J. Or, “A microcontroller based automatic sun tracker combined with a new solar energy conversion unit,” in *1998 International Conference on Power Electronic Drives and Energy Systems for Industrial Growth, 1998. Proceedings.*, vol. 1, 1998, pp. 488–492 Vol.1.

[17] Y. Away, A. Rahman, T. R. Auliandra, and M. Firdaus, “Performance Comparison Between PID and Fuzzy Algorithm for Sun Tracker Based on Tetrahedron Geometry Sensor,” in *2018 International Conference on Electrical Engineering and Informatics (ICELTICs)*, 2018, pp. 40–44.

[18] H. Allamehzadeh, “An Update on Solar Energy and Sun Tracker Technology with a Dual Axis Sun Tracker Application,” in *2019 IEEE 46th Photovoltaic Specialists Conference (PVSC)*, 2019, pp. 2037–2044.

[19] K. A. T. Djamiykov and I. Spasov, “Research and Design of Effective Positioning Algorithm for Solar Tracking System,” in *2018 IX National Conference with International Participation (ELECTRONICA)*, 2018, pp. 1–4.

[20] E. R. Zuhal and S. Marangozoglu, “New Design for Solar Panel Tracking System Based

on Solar Calculations," in *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2018, pp. 1042–1045.

[21] S. V. Mitrofanov, D. K. Baykasenov, and M. A. Suleev, "Simulation Model of Autonomous Solar Power Plant with Dual-Axis Solar Tracker," in *2018 International Ural Conference on Green Energy (UralCon)*, 2018, pp. 90–96.

[22] T.-S. Zhan, W.-M. Lin, M.-H. Tsai, and G.-S. Wang, "Design and Implementation of the Dual-Axis Solar Tracking System," in *2013 IEEE 37th Annual Computer Software and Applications Conference*, 2013, pp. 276–277.

[23] F. I. Mustafa, S. Shakir, F. F. Mustafa, and A. T. Naiyf, "Simple design and implementation of solar tracking system two axis with four sensors for Baghdad city," in *2018 9th International Renewable Energy Congress (IREC)*, 2018, pp. 1–5.

[24] M. Altayeb, S. Abdalla, and A. H. Mustafa, "Dual Axis Solar Sun Tracking System Based on GPS Satellite Receiver and Embedded System," in *2018 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, 2018, pp. 1–4.

[25] M. Singh, J. Singh, A. Garg, E. Sidhu, V. Singh, and A. Nag, "Efficient autonomous solar energy harvesting system utilizing dynamic offset feed mirrored parabolic dish integrated solar panel," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, 2016, pp. 1825–1829.

[26] I. Reda and A. Andreas, "Solar position algorithm for solar radiation applications," *Solar Energy*, vol. 76, no. 5, pp. 577–589, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038092X0300450X

[27] M. Jazayeri, S. Uysal, and K. Jazayeri, "A case study on solar data collection and effects of the sun's position in the sky on solar panel output characteristics in Northern Cyprus,"

in *2013 International Conference on Renewable Energy Research and Applications (ICR-ERA)*, 2013, pp. 184–189.

Appendix I

Source Code

I.1 Main Header File

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
```

```
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
```

```
#include "stm32l4xx_hal.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */

/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
```

```
/* USER CODE BEGIN EM */

/* USER CODE END EM */

void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define PR4_Pin GPIO_PIN_0
#define PR4_GPIO_Port GPIOC
#define PR3_Pin GPIO_PIN_1
#define PR3_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
```

```
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define PR1_Pin GPIO_PIN_4
#define PR1_GPIO_Port GPIOA
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
#define PR2_Pin GPIO_PIN_0
#define PR2_GPIO_Port GPIOB
#define GPS_FIX_Pin GPIO_PIN_8
#define GPS_FIX_GPIO_Port GPIOA
#define GPS_TX_Pin GPIO_PIN_10
#define GPS_TX_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif
```

```
#endif /* __MAIN_H */
```

Listing I.1: Main Header File

I.2 Main Solar Tracker Program

```
/* USER CODE BEGIN Header */
/**
  **********************************************************************
  * @file           : main.c
  * @brief          : Main program body
  **********************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  **********************************************************************
  */
/* USER CODE END Header */
```

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "MTK3339.h"
#include "PCA9685.h"
#include "SPA.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

```

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
//#define DEBUG_PRINT 1
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
DMA_HandleTypeDef hdma_adc1;
DMA_HandleTypeDef hdma_adc2;

I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
```

```c

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

uint32_t ADC_raw_values[4] = {0,0,0,0}; // raw ADC converted
    values
float    PR_values[4] = {0,0,0,0};    // ADC values converted
    to voltages
uint32_t solar_panel_raw = 0;      // raw solar panel ADC value
float    solar_panel_voltage = 0;   // Solar panel voltage

int programState = 0; // program state in state machine
int last_time = 0;     // temp variable to hold last GPS time
int isSunUp = 0;     // 0 if sun is down, 1 if sun is up

// enumerated states for FSM
enum States
{
Startup ,  // initialization , first GPS acquire
Running ,  // general operation throughout the day
Sleep    // no operation , checking for sunrise
};

```

```
// motor variables
    -------------------------------------------------------
typedef struct
{
  int UL;    // upper limit value for motor
  int LL;    // lower limit value for motor
  int pos;   // current position of motor
  int altitude; // calculated altitude of motor, translated
      from SPA
  int azimuth;  // calculated azimuth of motor, translated
      from SPA
} motor;

motor motor1 = {  // horizontal, uses azimuth
    .UL = 190,
    .LL = 10,
};

motor motor2 = {  // vertical, uses altitude
    .UL = 190,
    .LL = 10,
};

// gps variables
    ---------------------------------------------------------
```

```
extern struct GgaType gga;  // instance of GPS gga data
    structure
extern struct RmcType rmc;  // instance of GPS rmc data
    strcuture

// spa variables
    --------------------------------------------------------
spa_data spa = {  // instance of SPA data structure
  .delta_ut1 = 0,
  .delta_t = 37,
  .timezone = -4,
  .pressure = 2068.43,  // average annual pressure in
      Rochester, NY is 30in
  .temperature = 9.167, // average annual temperature is 48.5
      F
  .atmos_refract = 0.5667,
  .azm_rotation = 0,  // if i am in global village aiming at
      Salsarita's
  .slope = 0,
  .function = 1
};


/* USER CODE END PV */

```

```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
void PeriphCommonClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM3_Init(void);
static void MX_ADC2_Init(void);
/* USER CODE BEGIN PFP */
extern void GPS_Init();
extern void GPS_receive();
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

// function prototype for moving the motors, defined at bottom of file
int moveServo(int motor, int position);
```

```

/**
 * Use the ADC to read the solar panel voltage.
 * Convert it into scaled voltage value.
 */
void getSolarPanelVoltage( void )
{
  HAL_ADC_Start_DMA(&hadc2, &solar_panel_raw, 1);    // start
      ADC1 in DMA mode, point it to PR array
  HAL_Delay(2000);
   solar_panel_voltage = solar_panel_raw * (3.5 / 4096) * 2; //
       convert to voltage value
}

/**
 * Use the ADC to read the LDR values.
 * Convert them into voltage values.
 * Value decreases as light increases.
 */
void readLightSensors( void )
{
  HAL_ADC_Start_DMA(&hadc1, ADC_raw_values, 4);    // start
      ADC1 in DMA mode, point it to PR array
   for(int val=0; val<4; val++)
   {
```

```
    PR_values[val] = ADC_raw_values[val] * (3.5 / 4096);  // convert to voltage value
  }
}

/**
 * Check if the motor position is valid.
 * Return 0 if no correction is needed.
 * Otherwise, correct the position using LDRs
 * and return 1.
 */
int correctPosition( void )
{
  // first check if they are all equal, in which case no correction is needed
  if( (abs(PR_values[0] - PR_values[1]) < 0.5) &&
    (abs(PR_values[0] - PR_values[2]) < 0.5) &&
    (abs(PR_values[0] - PR_values[3]) < 0.5) )
  {
    return 0;
  }
  else
  {
    // correct the position
    int balanced = 0; // are LDR values roughly equal?
```

```

    while(!balanced)
    {
      readLightSensors();

      if( (abs(PR_values[0] - PR_values[1]) < 0.5) &&
          (abs(PR_values[0] - PR_values[2]) < 0.5) &&
          (abs(PR_values[0] - PR_values[3]) < 0.5) )
      {
        balanced = 1;
      }
      else
      {
        // average value of LDRs in each direction
        int top    = (ADC_raw_values[0] + ADC_raw_values[1]) /
           2;
        int bottom  = (ADC_raw_values[2] + ADC_raw_values[3])
           / 2;
        int left   = (ADC_raw_values[0] + ADC_raw_values[2]) /
           2;
        int right = (ADC_raw_values[1] + ADC_raw_values[3]) /
           2;

        if((top - bottom) > 0)
        {
```

```
          moveServo(1, motor1.pos + 2);
        }
        else if((top - bottom) < 0)
        {
          moveServo(1, motor1.pos - 2);
        }
        else if((left - right) > 0)
        {
          moveServo(2, motor1.pos - 2);
        }
        else if((left - right) < 0)
        {
          moveServo(1, motor1.pos + 2);
        }
      }
    }
    return 1;
  }
}

/**
 * Calculate motor angles from SPA results
 * spa: spa handle with the results
 */
void calculateMotorsPos(spa_data *sspa)
```

```
{
    //calculate servo value for solar azimuth
    int az = spa.azimuth; // azimuth adjusted to motor range
    int ze = spa.zenith;  // zenith adjusted to motor range

    if(az >= 0 && az < 90) // northeast
    {
        az = az + 90;
        motor1.azimuth = (int)(115 - ((az*97)/180));
        motor2.altitude = (int)(65 - ((ze*47.5)/90)); // bend
            backwards
    }
    else if(az >= 90 && az < 270) // south
    {
        az = az - 90;
        motor1.azimuth = (int)(115 - ((az*97)/180));
        motor2.altitude = (int)(65 + ((ze*47.5)/90)); // bend
            forwards
    }
    else if(az >= 270) // south
    {
        az = az - 270;
        motor1.azimuth = (int)(115 - ((az*97)/180));
        motor2.altitude = (int)(65 - ((ze*47.5)/90)); // bend
            backwards
```

```
    }
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration
     --------------------------------------------------------
     */

  /* Reset of all peripherals, Initializes the Flash interface
      and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

```

```
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

/* Configure the peripherals common clocks */
  PeriphCommonClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  MX_ADC1_Init();
  MX_TIM3_Init();
  MX_ADC2_Init();
  /* USER CODE BEGIN 2 */
  GPS_Init();
  /* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
    char str[100] = "System Start\r\n";
    HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str),
        HAL_MAX_DELAY);
    memset(&str, 0, sizeof(str));

    // Start Timer 3 and
    HAL_TIM_Base_Start(&htim3);
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1); // start servo 1
     CH1 PB4 D5 (18 to 115)
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2); // start servo 2
     CH2 PC7 D9 (18 to 113 is 180 degrees)

  // set each motor to default positions
  TIM3->CCR1 = 65;  // pointing due east
  TIM3->CCR2 = 65;  // poiting straight into the sky

  while (1)
  {
    // beginning of startup state
        -----------------------------------------------------
    while(programState == Startup)
    {
```

```

#ifdef DEBUG_PRINT
        sprintf(str, "GPS Query\r\n");
     HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str),
         HAL_MAX_DELAY);
     memset(&str, 0, sizeof(str));
#endif

        while (gga.fix == 0)  // query the GPS until we get
            valid GGA data
        {
          GPS_receive();  // get the data from the GPS module

#ifdef DEBUG_PRINT
        sprintf(str, "Time: %d : %d : %d\r\n"
                        "Latitude:    %0.4f%c\r\n"
                        "Longitude:   %0.4f%c\r\n"
                        "Fix:         %d\r\n"
                        "Satellites:  %d\r\n",
                  gga.hours, gga.minutes, gga.seconds,
                  gga.latitude_adj, gga.nsIndicator,
                  gga.longitude_adj, gga.ewIndicator,
                  gga.fix, gga.satellites);
        HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str),
            HAL_MAX_DELAY); // print the GPS data
```

```
        memset(&str, 0, sizeof(str));
#endif

        }
        while (rmc.status != 'A') // query the GPS until we get
            valid RMC data
        {
          GPS_receive();  // get the data from the GPS module
        }

        last_time = gga.minutes;
        // insert data to PCA calculator
        spa.longitude = getLongitudeAsDegrees();
        spa.latitude  = getLatitudeAsDegrees();
        spa.elevation = gga.altitude;
        spa.year    = rmc.year;
        spa.month   = rmc.month;
        spa.day     = rmc.day;
        spa.hour    = (gga.hours - 4); // subtract 4 for the UTC
            offset for current location
        spa.minute = gga.minutes;
        spa.second = gga.seconds;

        programState = Running;
```

```
} // end of startup state
    ---------------------------------------------------

// beginning of general operation state
    ---------------------------------------------------
while(programState == Running)
{
  // get new GPS data
  while(gga.minutes == last_time)
  {
    GPS_receive();
  }

  last_time = gga.minutes;
  // insert data to PCA calculator
  spa.longitude = getLongitudeAsDegrees();
  spa.latitude  = getLatitudeAsDegrees();
  spa.elevation = gga.altitude;
  spa.year    = rmc.year;
  spa.month   = rmc.month;
  spa.day     = rmc.day;
  spa.hour    = (gga.hours - 4); // subtract 4 for the UTC
     offset for current location
  spa.minute = gga.minutes;
  spa.second = gga.seconds;
```

```

        // run the SPA calculation
        if(spa_calculate(&spa) == 0)
        {

#ifdef DEBUG_PRINT
        sprintf(str, "SPA calculated\r\n");
        HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str), HAL_MAX_DELAY);
        memset(&str, 0, sizeof(str));
#endif

          // point the motors
          calculateMotorsPos(&spa); // calculate motor positions from SPA results

#ifdef DEBUG_PRINT
        sprintf(str, "motor 1 angle: %d\r\n"
                     "motor 2 angle: %d\r\n",
                  motor1.azimuth, motor2.altitude);
        HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str), HAL_MAX_DELAY);
        memset(&str, 0, sizeof(str));
#endif

```

```
            moveServo(1, motor1.azimuth);
            moveServo(2, motor2.altitude);

#ifdef DEBUG_PRINT
        sprintf(str, "Motors moved\r\n");
        HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str),
            HAL_MAX_DELAY);
        memset(&str, 0, sizeof(str));
#endif

            HAL_Delay(5000); // wait for 5 second for solar panel
                to get some sun

            getSolarPanelVoltage();
            // print solar panel voltage
            sprintf(str, "Time: %d : %d : %d\r\n"
                "SPA Output Voltage = %0.4f\r\n",(gga.hours - 4),
                    gga.minutes, gga.seconds, solar_panel_voltage);
            HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str)
                , HAL_MAX_DELAY);
            memset(&str, 0, sizeof(str));

            // then check the position with the LDRs
            readLightSensors();
            if(correctPosition() == 0)
```

```
        {
          sprintf(str, "Accurate\r\n");
          HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(
              str), HAL_MAX_DELAY);
          memset(&str, 0, sizeof(str));
        }
        else
        {
          sprintf(str, "NOT Accurate\r\n");
          HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(
              str), HAL_MAX_DELAY);
          memset(&str, 0, sizeof(str));
        }

        // check to see if the sun is still in the sky
        for(int ii=0; i<4; i++)
        {
          if(ADC_raw_values[ii] > 3000)
          {
            isSunUp = 0;
            programState = Sleep;
          }
        }

        // make solar panel flat
```

```
            moveServo(2, 65);

            HAL_Delay(5000); // wait for 1 second for solar panel
                to get some sun

            getSolarPanelVoltage();
            // print solar panel voltage
            sprintf(str, "Static Output Voltage = %0.4f\r\n",
                solar_panel_voltage);
            HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str)
                , HAL_MAX_DELAY);
            memset(&str, 0, sizeof(str));
        } // end of SPA operation

    HAL_Delay(10000); // wait 10 sec before reading again
    } // end of running state
        ----------------------------------------------------------

    // beginning of sleep state
        ----------------------------------------------------------
    while(programState == Sleep)
    {
        // sleep for 30 minutes
        HAL_Delay(180000);
        // check LDRs for sunrise
```

```
        readLightSensors();
        for(int ii=0; i<4; i++)
        {
          if(ADC_raw_values[ii] <= 3000)
          {
            isSunUp = 1;
            programState = Startup;
          }
        }

      } // end of sleep state
         -----------------------------------------------------

      /* USER CODE END WHILE */

      /* USER CODE BEGIN 3 */
    }
    /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
```

```
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(
      PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the RCC Oscillators according to the
      specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
      RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 10;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
```

```
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV8;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
}

```

```
/**
  * @brief Peripherals Common Clock Configuration
  * @retval None
  */
void PeriphCommonClock_Config(void)
{
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the peripherals clock
  */
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_ADC;
  PeriphClkInit.AdcClockSelection = RCC_ADCCLKSOURCE_PLLSAI1;
  PeriphClkInit.PLLSAI1.PLLSAI1Source = RCC_PLLSOURCE_HSI;
  PeriphClkInit.PLLSAI1.PLLSAI1M = 1;
  PeriphClkInit.PLLSAI1.PLLSAI1N = 8;
  PeriphClkInit.PLLSAI1.PLLSAI1P = RCC_PLLP_DIV7;
  PeriphClkInit.PLLSAI1.PLLSAI1Q = RCC_PLLQ_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1R = RCC_PLLR_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_ADC1CLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_MultiModeTypeDef multimode = {0};
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
```

```
hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
hadc1.Init.LowPowerAutoWait = DISABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.NbrOfConversion = 4;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc1.Init.DMAContinuousRequests = DISABLE;
hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
hadc1.Init.OversamplingMode = DISABLE;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
  Error_Handler();
}

/** Configure the ADC multi-mode
*/
multimode.Mode = ADC_MODE_INDEPENDENT;
if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
{
  Error_Handler();
}
```

```c

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_9;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_6CYCLES_5;
sConfig.SingleDiff = ADC_SINGLE_ENDED;
sConfig.OffsetNumber = ADC_OFFSET_NONE;
sConfig.Offset = 0;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
sConfig.Channel = ADC_CHANNEL_15;
sConfig.Rank = ADC_REGULAR_RANK_2;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
{
  Error_Handler();
}

/** Configure Regular Channel
*/
```

```
  sConfig.Channel = ADC_CHANNEL_2;
  sConfig.Rank = ADC_REGULAR_RANK_3;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = ADC_REGULAR_RANK_4;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief ADC2 Initialization Function
  * @param None
  * @retval None
```

```
  */
static void MX_ADC2_Init(void)
{

  /* USER CODE BEGIN ADC2_Init 0 */

  /* USER CODE END ADC2_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC2_Init 1 */

  /* USER CODE END ADC2_Init 1 */

  /** Common config
  */
  hadc2.Instance = ADC2;
  hadc2.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc2.Init.Resolution = ADC_RESOLUTION_12B;
  hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc2.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc2.Init.LowPowerAutoWait = DISABLE;
  hadc2.Init.ContinuousConvMode = DISABLE;
  hadc2.Init.NbrOfConversion = 1;
```

```
  hadc2.Init.DiscontinuousConvMode = DISABLE;
  hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc2.Init.ExternalTrigConvEdge =
     ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc2.Init.DMAContinuousRequests = DISABLE;
  hadc2.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  hadc2.Init.OversamplingMode = DISABLE;
  if (HAL_ADC_Init(&hadc2) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_16;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_6CYCLES_5;
  sConfig.SingleDiff = ADC_SINGLE_ENDED;
  sConfig.OffsetNumber = ADC_OFFSET_NONE;
  sConfig.Offset = 0;
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC2_Init 2 */
```

```

  /* USER CODE END ADC2_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00202538;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
```

```
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK)
{
  Error_Handler();
}

/** Configure Analogue filter
*/
if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1,
    I2C_ANALOGFILTER_ENABLE) != HAL_OK)
{
  Error_Handler();
}

/** Configure Digital filter
*/
if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
{
  Error_Handler();
}
/* USER CODE BEGIN I2C1_Init 2 */

```

```
  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
```

```
htim2.Init.Period = 4294967295;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload =
    TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &
    sMasterConfig) != HAL_OK)
{
  Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC,
    TIM_CHANNEL_1) != HAL_OK)
{
  Error_Handler();
}
/* USER CODE BEGIN TIM2_Init 2 */
```

```
  /* USER CODE END TIM2_Init 2 */
  HAL_TIM_MspPostInit(&htim2);

}

/**
  * @brief TIM3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM3_Init(void)
{

  /* USER CODE BEGIN TIM3_Init 0 */

  /* USER CODE END TIM3_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM3_Init 1 */

  /* USER CODE END TIM3_Init 1 */
```

```
htim3.Instance = TIM3;
htim3.Init.Prescaler = 400;
htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
htim3.Init.Period = 1000;
htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim3.Init.AutoReloadPreload =
    TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig)
    != HAL_OK)
{
  Error_Handler();
}
if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &
    sMasterConfig) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
      TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC,
      TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM3_Init 2 */

  /* USER CODE END TIM3_Init 2 */
  HAL_TIM_MspPostInit(&htim3);

}

/**
```

```
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 9600;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit =
      UART_ADVFEATURE_RXOVERRUNDISABLE_INIT;
```

```
  huart1.AdvancedInit.OverrunDisable =
      UART_ADVFEATURE_OVERRUN_DISABLE;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */
```

```

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT
      ;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * Enable DMA controller clock
```

```
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
  /* DMA1_Channel2_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel2_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOC_CLK_ENABLE();
__HAL_RCC_GPIOH_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin : B1_Pin */
GPIO_InitStruct.Pin = B1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : LD2_Pin */
GPIO_InitStruct.Pin = LD2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pin : GPS_FIX_Pin */
GPIO_InitStruct.Pin = GPS_FIX_Pin;
```

```
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPS_FIX_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/**
 * Move servo motor to a position.
 * motor:    the motor to be moved.
 * position: the position to move the motor to.
 * Return 0 if inputs are valid.
 * Return 1 if inputs are invalid.
 */
int moveServo(int motor, int position)
{
  if(motor == 1)
  {
    if(position < motor1.UL && position > motor1.LL) // if position is valid
    {
      TIM3->CCR1 = position;  // change PWM to position
      motor1.pos = position;  // update current motor position
      return 0; // no errors
```

```
    }
    else
    {
      return 1; // invalid position input
    }
  }
  else if(motor == 2)
  {
    if(position < motor2.UL && position > motor2.LL) // if
       position is valid
    {
      TIM3->CCR2 = position;  // change PWM to position
      motor2.pos = position;  // update current motor position
      return 0; // no errors
    }
    else
    {
      return 1; // invalid position input
    }
  }
  else
  {
    return 1; // invalid motor input
  }
}
```

```c






/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

```

```c
#ifdef USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Listing I.2: Main Source File

I.3 MTK3339 GPS Header File

```
#ifndef MTK3339_H
#define MTK3339_H

/**
 * An interface to the MTK3339 GPS module.
 */

/*
 * MTK3339.h
 *
 *  The original C++ code has been provided by
 *  EmbeddedArtists on mbed.com
 *  https://os.mbed.com/users/embeddedartists/code/MTK3339/
 *
 *  The following source file has been ported
 *  to C as well as modified to interact with the STM32
 *  platform.
 *
 *  Ported and expanded on: Mar 21, 2023
 *      Author: Austin Martinez
 */

    enum NmeaSentence
```

```
    {
            NmeaInvalid = 0,
            NmeaGga = 0x01,
//          NmeaGsa = 0x02,
//          NmeaGsv = 0x04,
//          NmeaRmc = 0x08,
            NmeaVtg = 0x10
        };

        struct GgaType {
            /** UTC time - hours */
            int hours;
            /** UTC time - minutes */
            int minutes;
            /** UTC time - seconds */
            int seconds;
            /** UTC time - milliseconds */
            int milliseconds;

            /** The latitude in ddmm.mmmm format (d = degrees, m =
                minutes) */
            double latitude;
            /** The longitude in dddmm.mmmm format */
            double longitude;
            /** North / South indicator */
```

```
        char nsIndicator;
        /** East / West indicator */
        char ewIndicator;
        /** Latitude adjusted to be in dd.mmmm format */
        double latitude_adj;
        /** Longitude adjusted to be in dd.mmmm format */
        double longitude_adj;

        /**
         * Position indicator:
         * 0 = Fix not available
         * 1 = GPS fix
         * 2 = Differential GPS fix
         */
        int fix;

        /** Number of used satellites */
        int satellites;
        /** Horizontal Dilution of Precision */
        double hdop;
        /** antenna altitude above/below mean sea-level */
        double altitude;
        /** geoidal separation */
        double geoidal;
    };
```

```
    struct RmcType {
        /** UTC time - hours */
        int hours;
        /** UTC time - minutes */
        int minutes;
        /** UTC time - seconds */
        int seconds;
        /** UTC time - milliseconds */
        int milliseconds;

        /** Status Indicator:
         * A = data valid
         * V = data NOT valid
         */
        char status;

        /** The latitude in ddmm.mmmm format (d = degrees, m =
            minutes) */
        double latitude;
        /** The longitude in dddmm.mmmm format */
        double longitude;
        /** North / South indicator */
        char nsIndicator;
```

```
        /** East / West indicator */
        char ewIndicator;
        /** Speed over ground - knots */
        double speed;
        /** Course over ground - knots */
        double course;
        /** UTC Date - year */
        int year;
        /** UTC Date - month */
        int month;
        /** UTC Date - day */
        int day;

    };


    struct VtgType {
        /** heading in degrees */
        double course;
        /** speed in Knots */
        double speedKnots;
        /** Speed in kilometer  per hour */
        double speedKmHour;
        /**
         * Mode
```

```
         * A = Autonomous mode
         * D = Differential mode
         * E = Estimated mode
         */
        char mode;
    };

    /**
     * Get latitude in degrees (decimal format)
     */
    double getLatitudeAsDegrees();
    /**
     * Get longitude in degrees (decimal format)
     */
    double getLongitudeAsDegrees();

    /**
     * Time, position and fix related data
     */
//      struct GgaType gga;

    /**
     * Course and speed information relative to ground
     */
//      struct VtgType vtg;
```

```
    enum PrivConstants
  {
        MTK3339_BUF_SZ = 255
    };

    enum DataState
  {
        StateStart = 0,
        StateData
    };

//      char _buf[MTK3339_BUF_SZ];
//    int _bufPos;
//    enum DataState _state;

  void GPS_Init( void );
    void parseGGA(char* data, int dataLen);
    void parseRMC(char* data, int dataLen);
    void parseVTG(char* data, int dataLen);
    void parseData(char* data, int len);
    void GPS_receive( void );
```

```

#endif
```

Listing I.3: MTK3339 GPS Header File

I.4 MTK3339 GPS Source File

```c

/*
 * MTK3339.c
 *
 *  The original C++ code has been provided by
 *  EmbeddedArtists on mbed.com
 *  https://os.mbed.com/users/embeddedartists/code/MTK3339/
 *
 *  The following source file has been ported
 *  to C as well as modified to interact with the STM32
 *  platform.
 *
 *  Ported and expanded on: Mar 21, 2023
 *      Author: Austin Martinez
 */

#include "main.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "MTK3339.h"

extern UART_HandleTypeDef huart1; // UART for GPS
```

```
extern UART_HandleTypeDef huart2; // UART for printing to console

char _buf[MTK3339_BUF_SZ];
int _bufPos;
enum DataState _state;
struct GgaType gga;
struct VtgType vtg;
struct RmcType rmc;

void GPS_Init()
{
    memset(&gga, 0, sizeof(gga));
    memset(&vtg, 0, sizeof(vtg));
}

double getLatitudeAsDegrees()
{
    if(gga.fix == 0 || gga.nsIndicator == 0) return 0;

    double l = gga.latitude;
    char ns = gga.nsIndicator;

    // convert from ddmm.mmmm to degrees only
    // 60 minutes is 1 degree
```

```
    int deg = (int)(l / 100);
    l = (l - deg*100.0) / 60.0;
    l = deg + l;
    if (ns == 'S') l = -l;

    return l;
}

double getLongitudeAsDegrees()
{
    if(gga.fix == 0 || gga.ewIndicator == 0) return 0;

    double l = gga.longitude;
    char ew = gga.ewIndicator;

    // convert from ddmm.mmmm to degrees only
    // 60 minutes is 1 degree

    int deg = (int)(l / 100);
    l = (l - deg*100) / 60;
    l = deg + l;
    if (ew == 'W') l = -l;

    return l;
```

```
}

void parseGGA(char* data, int dataLen)
{
    //http://aprs.gids.nl/nmea/#gga

    double tm = 0;

    memset(&gga, 0, sizeof(gga));

    char* p = data;
    int pos = 0;

    p = strchr(p, ',');
    while (p != NULL && *p != 0) {
        p++;

        switch(pos) {
            case 0: // time: hhmmss.sss
                tm = strtod(p, NULL);
                gga.hours = (int)(tm / 10000);
                gga.minutes = ((int)tm % 10000) / 100;
                gga.seconds = ((int)tm % 100);
                gga.milliseconds = (int)(tm * 1000) % 1000;
                break;
```

```
            case 1: // latitude: ddmm.mmmm
                gga.latitude = strtod(p, NULL);
                gga.latitude_adj = gga.latitude / 100;
                break;
            case 2: // N/S indicator (north or south)
                if (*p == 'N' || *p == 'S') {
                    gga.nsIndicator = *p;
                }
                break;
            case 3: // longitude: dddmm.mmmm
                gga.longitude = strtod(p, NULL);
                gga.longitude_adj = gga.longitude / 100;
                break;
            case 4: // E/W indicator (east or west)
                if (*p == 'E' || *p == 'W') {
                    gga.ewIndicator = *p;
                }
                break;
            case 5: // position indicator (1=no fix, 2=GPS fix, 3=Differential)
                gga.fix = strtol(p, NULL, 10);
                break;
            case 6: // num satellites
                gga.satellites = strtol(p, NULL, 10);
                break;
```

```
                case 7: // hdop
                    gga.hdop = strtod(p, NULL);
                    break;
                case 8: // altitude
                    gga.altitude = strtod(p, NULL);
                    break;
                case 9: // units
                    // ignore units
                    break;
                case 10: // geoidal separation
                    gga.geoidal = strtod(p, NULL);
                    break;
            }
            pos++;

            p = strchr(p, ',');
        }

}

void parseRMC(char* data, int dataLen)
{
    //http://aprs.gids.nl/nmea/#gga

    double tm = 0;  // time
```

```
    double dt = 0;  // date

    memset(&rmc, 0, sizeof(rmc));

    char* p = data;
    int pos = 0;

    p = strchr(p, ',');
    while (p != NULL && *p != 0) {
        p++;

        switch(pos) {
            case 0: // time: hhmmss.sss
                tm = strtod(p, NULL);
                rmc.hours = (int)(tm / 10000);
                rmc.minutes = ((int)tm % 10000) / 100;
                rmc.seconds = ((int)tm % 100);
                rmc.milliseconds = (int)(tm * 1000) % 1000;
                break;
            case 1: // status: A or V
              if (*p == 'A' || *p == 'V') {
                rmc.status = *p;
              }
              break;
            case 2: // latitude: ddmm.mmmm
```

```
            rmc.latitude = strtod(p, NULL);
              break;
          case 3: // N/S indicator (north or south)
              if (*p == 'N' || *p == 'S') {
                rmc.nsIndicator = *p;
              }
              break;
          case 4: // longitude: dddmm.mmmm
            rmc.longitude = strtod(p, NULL);
              break;
          case 5: // E/W indicator (east or west)
              if (*p == 'E' || *p == 'W') {
                rmc.ewIndicator = *p;
              }
              break;
          case 6: // speed: z.z
            rmc.speed = strtod(p, NULL);
              break;
          case 7: // course: y.y
              rmc.course = strtod(p, NULL);
              break;
          case 8: // UTC date: ddmmyy
              dt = strtod(p, NULL);
              rmc.day = (int)(dt / 10000);
    rmc.month = ((int)dt % 10000) / 100;
```

```
            rmc.year = ((int)dt % 100);
            rmc.year = rmc.year + 2000;
                    break;
            }
            pos++;

            p = strchr(p, ',');
        }

}

void parseVTG(char* data, int dataLen)
{


        char* p = data;
        int pos = 0;

        memset(&vtg, 0, sizeof(vtg));

        p = strchr(p, ',');
        while (p != NULL && *p != 0) {
            p++;

            switch(pos) {
```

```
            case 0: // course in degrees
                vtg.course = strtod(p, NULL);
                break;
            case 1: // Reference (T)
                break;
            case 2: // course magnetic (need customization)
                break;
            case 3: // reference (M)
                break;
            case 4: // speed in knots
                vtg.speedKnots = strtod(p, NULL);
                break;
            case 5: // units (N)
                break;
            case 6: // speed in Km/h
                vtg.speedKmHour = strtod(p, NULL);
                break;
            case 7: // units (K)
                break;
            case 8: // mode
                if (*p == 'A' || *p == 'D' || *p == 'E') {
                    vtg.mode = *p;
                }

                break;
```

```
            }

            pos++;

            p = strchr(p, ',');
        }
}

void parseData(char* data, int len)
{
        do {

            // verify checksum
            if (len < 3 || (len > 3 && data[len-3] != '*'))
            {
                // invalid data
                break;
            }
            int sum = strtol(&data[len-2], NULL, 16);
            for(int i = 1; i < len-3; i++)
            {
                sum ^= data[i];
            }
            if (sum != 0)
```

```
            {
                // invalid checksum
                break;
            }


            if (strncmp("$GPGGA", data, 6) == 0)
            {
                parseGGA(data, len);
            }
            else if (strncmp("$GPVTG", data, 6) == 0)
            {
                parseVTG(data, len);
            }


        } while(0);
}

void GPS_receive()
{
  uint8_t rxByte = 0;
  char rx_buffer[80];
  int rx_idx = 0;
  _Bool data_received = 0;
```

```
  _Bool GGA_parsed = 0;
  _Bool RMC_parsed = 0;

  while (!GGA_parsed && !RMC_parsed)
  {
    // receive the UART data from the GPS
    // wait until we see $GPGGA
    if ((rx_idx < 80) && !data_received)
    {
      HAL_UART_Receive(&huart1, &rxByte, 1, HAL_MAX_DELAY);
      rx_buffer[rx_idx] = rxByte;
      if (rx_buffer[rx_idx] == '$') // beginning of some GPS output
      {
        rx_idx++;
      }
      else if((rx_idx > 0) && (rx_buffer[rx_idx] != '*' || rx_buffer[rx_idx] != '\r'))  // if GPS data has started AND we aren't at the end, keep collecting it
      {
        rx_idx++;
      }
      else if(rx_buffer[rx_idx] == '*' || rx_buffer[rx_idx] == '\r')  // if we are at the end of GPS data, data has been received
```

```
            {
              data_received = 1;
            }
            else     // if not beginning of GPS data ($), reset to
                beginning of buffer
            {
              rx_idx = 0;
            }
          }
          // once data is received or the buffer has run out room,
              check if it is GGA data or RMC data
          else
          {
            data_received = 0; // reset data received

            if (strncmp("$GPGGA", rx_buffer, 6) == 0)    // if it is
                GGA data
            {
//            char str[30] = "$GPGGA found\r\n";
//            HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(
    str), HAL_MAX_DELAY);
              parseGGA(rx_buffer, rx_idx);          // if it is GGA,
                  parse it
              memset(&rx_buffer, 0, sizeof(rx_buffer));    // clear
                  the RX buffer
```

```
          rx_idx = 0;                    // reset UART RX buffer index
          GGA_parsed = 1;
        }
        else if (strncmp("$GPRMC", rx_buffer, 6) == 0) // if it is RMC data
        {
//          char str[30] = "$GPRMC found\r\n";
//          HAL_UART_Transmit(&huart2, (uint8_t *)str, sizeof(str), HAL_MAX_DELAY);
          parseRMC(rx_buffer, rx_idx);          // if it is GGA, parse it
          memset(&rx_buffer, 0, sizeof(rx_buffer));    // clear the RX buffer
          rx_idx = 0;                    // reset UART RX buffer index
          RMC_parsed = 1;
        }
        else
        {
          memset(&rx_buffer, 0, sizeof(rx_buffer)); // clear the RX buffer
          rx_idx = 0; // reset UART RX buffer index
        }
      }
```

```
    }
}
```

Listing I.4: MTK3339 GPS Source File

I.5 SPA Header File

```
#ifndef INC_SPA_H_
#define INC_SPA_H_



///////////////////////////////////////////////
//            HEADER FILE for SPA.C          //
//                                           //
//      Solar Position Algorithm (SPA)       //
//                   for                     //
//        Solar Radiation Application        //
//                                           //
//               May 12, 2003                //
//                                           //
//   Filename: SPA.H                         //
//                                           //
//   Afshin Michael Andreas                  //
//   afshin_andreas@nrel.gov (303)384-6383   //
//                                           //
//   Measurement & Instrumentation Team      //
//   Solar Radiation Research Laboratory     //
//   National Renewable Energy Laboratory    //
//   1617 Cole Blvd, Golden, CO 80401        //
```

```
// ///////////////////////////////////////////

// ////////////////////////////////////////////////////////////////////
//                                                                    //
// Usage:                                                             //
//                                                                    //
//    1) In calling program, include this header file,                //
//       by adding this line to the top of file:                      //
//           #include "spa.h"                                         //
//                                                                    //
//    2) In calling program, declare the SPA structure:               //
```

```
//              spa_data spa;                                          //
//                                                                     //
//   3) Enter the required input values into SPA structure             //
//      (input values listed in comments below)                        //
//                                                                     //
//   4) Call the SPA calculate function and pass the SPA structure     //
//      (prototype is declared at the end of this header file):        //
//              spa_calculate(&spa);                                   //
//                                                                     //
//   Selected output values (listed in comments below) will be         //
//   computed and returned in the passed SPA structure.  Output        //
```

```
//    will based on function code selected from enumeration below.       //
//                                                                       //
//    Note: A non-zero return code from spa_calculate() indicates that //
//          one of the input values did not pass simple bounds tests.  //
//          The valid input ranges and return error codes are also      //
//          listed below.                                               //
//                                                                       //
// ////////////////////////////////////////////////////////////////////////////////


//enumeration for function codes to select desired final outputs from SPA
enum {
    SPA_ZA,             //calculate zenith and azimuth
```

```
    SPA_ZA_INC,         //calculate zenith, azimuth, and incidence
    SPA_ZA_RTS,         //calculate zenith, azimuth, and sun rise/transit/set values
    SPA_ALL,            //calculate all SPA output values
};

typedef struct
{
    //----------------------INPUT VALUES------------------------

    int year;            // 4-digit year,      valid range: -2000 to 6000, error code: 1
    int month;           // 2-digit month,         valid range: 1 to  12,  error code: 2
    int day;             // 2-digit day,           valid range: 1 to  31,  error code: 3
    int hour;            // Observer local hour,   valid range: 0 to  24,  error code: 4
    int minute;          // Observer local minute, valid range: 0 to  59,  error code: 5
    double second;       // Observer local second, valid range: 0 to <60,  error code: 6

```

```
double delta_ut1;    // Fractional second difference between UTC and UT which is used
                     // to adjust UTC for earth's irregular rotation rate and is derived
                     // from observation only and is reported in this bulletin:
                     // http://maia.usno.navy.mil/ser7/ser7.dat,
                     // where delta_ut1 = DUT1
                     // valid range: -1 to 1 second (exclusive), error code 17

  double delta_t;      // Difference between earth rotation time and terrestrial time
                       // It is derived from observation only and is reported in this
                       // bulletin: http://maia.usno.navy.mil/ser7/ser7.dat,
                       // where delta_t = 32.184 + (TAI-UTC) - DUT1
                       // valid range: -8000 to 8000 seconds, error code: 7

  double timezone;     // Observer time zone (negative west of Greenwich)
```

```
                         // valid range: -18   to   18 hours, error code: 8

    double longitude;    // Observer longitude (negative west of Greenwich)
                         // valid range: -180  to  180 degrees, error code: 9

    double latitude;     // Observer latitude (negative south of equator)
                         // valid range: -90   to   90 degrees, error code: 10

    double elevation;    // Observer elevation [meters]
                         // valid range: -6500000 or higher meters,    error code: 11

    double pressure;     // Annual average local pressure [millibars]
                         // valid range:    0 to 5000 millibars,       error code: 12

    double temperature;  // Annual average local temperature [degrees Celsius]
```

```
                            // valid range: -273 to 6000 degrees Celsius, error code; 13

    double slope;           // Surface slope (measured from the horizontal plane)
                            // valid range: -360 to 360 degrees, error code: 14

    double azm_rotation;    // Surface azimuth rotation (measured from south to projection of
                            //     surface normal on horizontal plane, negative east)
                            // valid range: -360 to 360 degrees, error code: 15

    double atmos_refract;// Atmospheric refraction at sunrise and sunset (0.5667 deg is typical)
                            // valid range: -5   to   5 degrees, error code: 16

    int function;           // Switch to choose functions for desired output (from enumeration)

    //-----------------Intermediate OUTPUT VALUES--------------------
```

```
    double jd;          //Julian day
    double jc;          //Julian century

    double jde;         //Julian ephemeris day
    double jce;         //Julian ephemeris century
    double jme;         //Julian ephemeris millennium

    double l;           //earth heliocentric longitude [degrees]
    double b;           //earth heliocentric latitude [degrees]
    double r;           //earth radius vector [Astronomical Units, AU]

    double theta;       //geocentric longitude [degrees]
    double beta;        //geocentric latitude [degrees]

    double x0;          //mean elongation (moon-sun) [degrees]
    double x1;          //mean anomaly (sun) [degrees]
    double x2;          //mean anomaly (moon) [degrees]
    double x3;          //argument latitude (moon) [degrees]
    double x4;          //ascending longitude (moon) [degrees]

    double del_psi;     //nutation longitude [degrees]
```

```
    double del_epsilon; //nutation obliquity [degrees]
    double epsilon0;    //ecliptic mean obliquity [arc seconds]
    double epsilon;     //ecliptic true obliquity  [degrees]

    double del_tau;     //aberration correction [degrees]
    double lamda;       //apparent sun longitude [degrees]
    double nu0;         //Greenwich mean sidereal time [degrees]
    double nu;          //Greenwich sidereal time [degrees]

    double alpha;       //geocentric sun right ascension [degrees]
    double delta;       //geocentric sun declination [degrees]

    double h;           //observer hour angle [degrees]
    double xi;          //sun equatorial horizontal parallax [degrees]
    double del_alpha;   //sun right ascension parallax [degrees]
    double delta_prime; //topocentric sun declination [degrees]
    double alpha_prime; //topocentric sun right ascension [degrees]
```

```
    double h_prime;        //topocentric local hour angle [degrees]

    double e0;             //topocentric elevation angle (uncorrected) [degrees]
    double del_e;          //atmospheric refraction correction [degrees]
    double e;              //topocentric elevation angle (corrected) [degrees]

    double eot;            //equation of time [minutes]
    double srha;           //sunrise hour angle [degrees]
    double ssha;           //sunset hour angle [degrees]
    double sta;            //sun transit altitude [degrees]

    //---------------------Final OUTPUT VALUES------------------------

    double zenith;        //topocentric zenith angle [degrees]
    double azimuth_astro; //topocentric azimuth angle (westward from south) [for astronomers]
    double azimuth;       //topocentric azimuth angle (eastward from north) [for navigators and solar radiation]
    double incidence;     //surface incidence angle [degrees]

```

```c
    double suntransit;   //local sun transit time (or solar noon) [fractional hour]
    double sunrise;      //local sunrise time (+/- 30 seconds) [fractional hour]
    double sunset;       //local sunset time (+/- 30 seconds) [fractional hour]

} spa_data;

//-------------- Utility functions for other applications (such as NREL's SAMPA) --------------
double deg2rad(double degrees);
double rad2deg(double radians);
double limit_degrees(double degrees);
double third_order_polynomial(double a, double b, double c, double d, double x);
double geocentric_right_ascension(double lamda, double epsilon, double beta);
double geocentric_declination(double beta, double epsilon, double lamda);
double observer_hour_angle(double nu, double longitude, double alpha_deg);
void   right_ascension_parallax_and_topocentric_dec(double latitude, double elevation,
```

```
              double xi, double h, double delta, double *
                  delta_alpha, double *delta_prime);
double topocentric_right_ascension(double alpha_deg, double
    delta_alpha);
double topocentric_local_hour_angle(double h, double
    delta_alpha);
double topocentric_elevation_angle(double latitude, double
    delta_prime, double h_prime);
double atmospheric_refraction_correction(double pressure,
    double temperature,
                                                    double atmos_refract,
                                                        double e0);
double topocentric_elevation_angle_corrected(double e0, double
     delta_e);
double topocentric_zenith_angle(double e);
double topocentric_azimuth_angle_astro(double h_prime, double
    latitude, double delta_prime);
double topocentric_azimuth_angle(double azimuth_astro);


//Calculate SPA output values (in structure) based on input
    values passed in structure
int spa_calculate(spa_data *spa);


```

```
#endif /* INC_SPA_H_ */
```

Listing I.5: SPA Header File

I.6 SPA Source File

```
/////////////////////////////////////////////
//      Solar Position Algorithm (SPA)     //
//                   for                   //
//        Solar Radiation Application      //
//                                         //
//               May 12, 2003              //
//                                         //
//   Filename: SPA.C                       //
//                                         //
//   Afshin Michael Andreas                //
//   Afshin.Andreas@NREL.gov (303)384-6383 //
//                                         //
//   Metrology Laboratory                  //
//   Solar Radiation Research Laboratory   //
//   National Renewable Energy Laboratory  //
//   15013 Denver W Pkwy, Golden, CO 80401 //
/////////////////////////////////////////////

/////////////////////////////////////////////
//   See the SPA.H header file for usage   //
//                                         //
//   This code is based on the NREL        //
//   technical report "Solar Position      //
```

```
//    Algorithm for Solar Radiation          //
//    Application" by I. Reda & A. Andreas   //
// ///////////////////////////////////////////

// ////////////////////////////////////////////////////////////////////////////
//
//    NOTICE
//    Copyright (C) 2008-2011 Alliance for Sustainable Energy, LLC, All Rights Reserved
//
//The Solar Position Algorithm ("Software") is code in development prepared by employees of the
//Alliance for Sustainable Energy, LLC, (hereinafter the "Contractor"), under Contract No.
//DE-AC36-08GO28308 ("Contract") with the U.S. Department of Energy (the "DOE"). The United
//States Government has been granted for itself and others acting on its behalf a paid-up, non-
//exclusive, irrevocable, worldwide license in the Software to reproduce, prepare derivative
//works, and perform publicly and display publicly. Beginning five (5) years after the date
```

```
//permission to assert copyright is obtained from the DOE, and subject to any subsequent five
//(5) year renewals, the United States Government is granted for itself and others acting on
//its behalf a paid-up, non-exclusive, irrevocable, worldwide license in the Software to
//reproduce, prepare derivative works, distribute copies to the public, perform publicly and
//display publicly, and to permit others to do so. If the Contractor ceases to make this
//computer software available, it may be obtained from DOE's Office of Scientific and Technical
//Information's Energy Science and Technology Software Center (ESTSC) at P.O. Box 1020, Oak
//Ridge, TN 37831-1020. THIS SOFTWARE IS PROVIDED BY THE CONTRACTOR "AS IS" AND ANY EXPRESS OR
//IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
//AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CONTRACTOR OR THE
//U.S. GOVERNMENT BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
//WHATSOEVER, INCLUDING BUT NOT LIMITED TO CLAIMS ASSOCIATED WITH THE LOSS OF DATA OR PROFITS,
```

```
//WHICH MAY RESULT FROM AN ACTION IN CONTRACT, NEGLIGENCE OR OTHER TORTIOUS CLAIM THAT ARISES
//OUT OF OR IN CONNECTION WITH THE ACCESS, USE OR PERFORMANCE OF THIS SOFTWARE.
//
//The Software is being provided for internal, noncommercial purposes only and shall not be
//re-distributed. Please contact the NREL Commercialization and Technology Transfer Office
//for information concerning a commercial license to use the Software, visit:
//http://midcdmz.nrel.gov/spa/ for the contact information.
//
//As a condition of using the Software in an application, the developer of the application
//agrees to reference the use of the Software and make this Notice readily accessible to any
//end-user in a Help|About screen or equivalent manner.
//
// ///////////////////////////////////////////////////////////////////////////

// ///////////////////////////////////////////////////////////////////////////
```

```
// Revised 27-FEB-2004 Andreas
//         Added bounds check on inputs and return value for spa_calculate().
// Revised 10-MAY-2004 Andreas
//         Changed temperature bound check minimum from -273.15 to -273 degrees C.
// Revised 17-JUN-2004 Andreas
//         Corrected a problem that caused a bogus sunrise/set/transit on the equinox.
// Revised 18-JUN-2004 Andreas
//         Added a "function" input variable that allows the selecting of desired outputs.
// Revised 21-JUN-2004 Andreas
//         Added 3 new intermediate output values to SPA structure (srha, ssha, & sta).
// Revised 23-JUN-2004 Andreas
//         Enumerations for "function" were renamed and 2 were added.
//         Prevented bound checks on inputs that are not used (based on function).
// Revised 01-SEP-2004 Andreas
//         Changed a local variable from integer to double.
// Revised 12-JUL-2005 Andreas
```

```
//            Put a limit on the EOT calculation, so that the result is between -20 and 20.
// Revised 26-OCT-2005 Andreas
//            Set the atmos. refraction correction to zero, when sun is below horizon.
//            Made atmos_refract input a requirement for all "functions".
//            Changed atmos_refract bound check from +/- 10 to +/- 5 degrees.
// Revised 07-NOV-2006 Andreas
//            Corrected 3 earth periodic terms in the L_TERMS array.
//            Corrected 2 earth periodic terms in the R_TERMS array.
// Revised 10-NOV-2006 Andreas
//            Corrected a constant used to calculate topocentric sun declination.
//            Put a limit on observer hour angle, so result is between 0 and 360.
// Revised 13-NOV-2006 Andreas
//            Corrected calculation of topocentric sun declination.
//            Converted all floating point inputs in spa structure to doubles.
// Revised 27-FEB-2007 Andreas
```

```
//            Minor correction made as to when atmos. refraction correction is set to zero.
// Revised 21-JAN-2008 Andreas
//            Minor change to two variable declarations.
// Revised 12-JAN-2009 Andreas
//            Changed timezone bound check from +/-12 to +/-18 hours.
// Revised 14-JAN-2009 Andreas
//            Corrected a constant used to calculate ecliptic mean obliquity.
// Revised 01-APR-2013 Andreas
//          Replace floor with new integer function for tech. report consistency, no affect on results.
//            Add "utility" function prototypes to header file for use with NREL's SAMPA.
//            Rename 4 "utility" function names (remove "sun") for clarity with NREL's SAMPA.
//          Added delta_ut1 as required input, which the fractional second difference between UT and UTC.
//            Time must be input w/o delta_ut1 adjustment, instead of assuming adjustment was pre-applied.
// Revised 10-JUL-2014 Andreas
//            Change second in spa_data structure from an integer to double to allow fractional second
// Revised 08-SEP-2014 Andreas
```

```
//          Corrected description of azm_rotation in header file
//          Limited azimuth180 to range of 0 to 360 deg (instead of -180 to 180) for tech report consistency
//          Changed all variables names from azimuth180 to azimuth_astro
//          Renamed 2 "utility" function names for consistency
// ///////////////////////////////////////////////////////////////////////

#include <math.h>
#include "spa.h"

#define PI         3.1415926535897932384626433832795028841971
#define SUN_RADIUS 0.26667

#define L_COUNT 6
#define B_COUNT 2
#define R_COUNT 5
#define Y_COUNT 63

#define L_MAX_SUBCOUNT 64
#define B_MAX_SUBCOUNT 5
#define R_MAX_SUBCOUNT 40
```

```
enum {TERM_A, TERM_B, TERM_C, TERM_COUNT};
enum {TERM_X0, TERM_X1, TERM_X2, TERM_X3, TERM_X4,
    TERM_X_COUNT};
enum {TERM_PSI_A, TERM_PSI_B, TERM_EPS_C, TERM_EPS_D,
    TERM_PE_COUNT};
enum {JD_MINUS, JD_ZERO, JD_PLUS, JD_COUNT};
enum {SUN_TRANSIT, SUN_RISE, SUN_SET, SUN_COUNT};

#define TERM_Y_COUNT TERM_X_COUNT

const int l_subcount[L_COUNT] = {64,34,20,7,3,1};
const int b_subcount[B_COUNT] = {5,2};
const int r_subcount[R_COUNT] = {40,10,6,2,1};

///////////////////////////////////////////////////////
///   Earth Periodic Terms
///////////////////////////////////////////////////////
const double L_TERMS[L_COUNT][L_MAX_SUBCOUNT][TERM_COUNT]=
{
    {
        {175347046.0,0,0},
        {3341656.0,4.6692568,6283.07585},
        {34894.0,4.6261,12566.1517},
        {3497.0,2.7441,5753.3849},
```

```
{3418.0,2.8289,3.5231},
{3136.0,3.6277,77713.7715},
{2676.0,4.4181,7860.4194},
{2343.0,6.1352,3930.2097},
{1324.0,0.7425,11506.7698},
{1273.0,2.0371,529.691},
{1199.0,1.1096,1577.3435},
{990,5.233,5884.927},
{902,2.045,26.298},
{857,3.508,398.149},
{780,1.179,5223.694},
{753,2.533,5507.553},
{505,4.583,18849.228},
{492,4.205,775.523},
{357,2.92,0.067},
{317,5.849,11790.629},
{284,1.899,796.298},
{271,0.315,10977.079},
{243,0.345,5486.778},
{206,4.806,2544.314},
{205,1.869,5573.143},
{202,2.458,6069.777},
{156,0.833,213.299},
{132,3.411,2942.463},
{126,1.083,20.775},
```

```
{115,0.645,0.98},
{103,0.636,4694.003},
{102,0.976,15720.839},
{102,4.267,7.114},
{99,6.21,2146.17},
{98,0.68,155.42},
{86,5.98,161000.69},
{85,1.3,6275.96},
{85,3.67,71430.7},
{80,1.81,17260.15},
{79,3.04,12036.46},
{75,1.76,5088.63},
{74,3.5,3154.69},
{74,4.68,801.82},
{70,0.83,9437.76},
{62,3.98,8827.39},
{61,1.82,7084.9},
{57,2.78,6286.6},
{56,4.39,14143.5},
{56,3.47,6279.55},
{52,0.19,12139.55},
{52,1.33,1748.02},
{51,0.28,5856.48},
{49,0.49,1194.45},
{41,5.37,8429.24},
```

```
        {41,2.4,19651.05},
        {39,6.17,10447.39},
        {37,6.04,10213.29},
        {37,2.57,1059.38},
        {36,1.71,2352.87},
        {36,1.78,6812.77},
        {33,0.59,17789.85},
        {30,0.44,83996.85},
        {30,2.74,1349.87},
        {25,3.16,4690.48}
    },
    {
        {628331966747.0,0,0},
        {206059.0,2.678235,6283.07585},
        {4303.0,2.6351,12566.1517},
        {425.0,1.59,3.523},
        {119.0,5.796,26.298},
        {109.0,2.966,1577.344},
        {93,2.59,18849.23},
        {72,1.14,529.69},
        {68,1.87,398.15},
        {67,4.41,5507.55},
        {59,2.89,5223.69},
        {56,2.17,155.42},
        {45,0.4,796.3},
```

```
        {36,0.47,775.52},
        {29,2.65,7.11},
        {21,5.34,0.98},
        {19,1.85,5486.78},
        {19,4.97,213.3},
        {17,2.99,6275.96},
        {16,0.03,2544.31},
        {16,1.43,2146.17},
        {15,1.21,10977.08},
        {12,2.83,1748.02},
        {12,3.26,5088.63},
        {12,5.27,1194.45},
        {12,2.08,4694},
        {11,0.77,553.57},
        {10,1.3,6286.6},
        {10,4.24,1349.87},
        {9,2.7,242.73},
        {9,5.64,951.72},
        {8,5.3,2352.87},
        {6,2.65,9437.76},
        {6,4.67,4690.48}
    },
    {
        {52919.0,0,0},
        {8720.0,1.0721,6283.0758},
```

```
        {309.0,0.867,12566.152},
        {27,0.05,3.52},
        {16,5.19,26.3},
        {16,3.68,155.42},
        {10,0.76,18849.23},
        {9,2.06,77713.77},
        {7,0.83,775.52},
        {5,4.66,1577.34},
        {4,1.03,7.11},
        {4,3.44,5573.14},
        {3,5.14,796.3},
        {3,6.05,5507.55},
        {3,1.19,242.73},
        {3,6.12,529.69},
        {3,0.31,398.15},
        {3,2.28,553.57},
        {2,4.38,5223.69},
        {2,3.75,0.98}
    },
    {
        {289.0,5.844,6283.076},
        {35,0,0},
        {17,5.49,12566.15},
        {3,5.2,155.42},
        {1,4.72,3.52},
```

```
            {1,5.3,18849.23},
            {1,5.97,242.73}
        },
        {
            {114.0,3.142,0},
            {8,4.13,6283.08},
            {1,3.84,12566.15}
        },
        {
            {1,3.14,0}
        }
};

const double B_TERMS[B_COUNT][B_MAX_SUBCOUNT][TERM_COUNT]=
{
        {
            {280.0,3.199,84334.662},
            {102.0,5.422,5507.553},
            {80,3.88,5223.69},
            {44,3.7,2352.87},
            {32,4,1577.34}
        },
        {
            {9,3.9,5507.55},
            {6,1.73,5223.69}
```

```
    }
};

const double R_TERMS[R_COUNT][R_MAX_SUBCOUNT][TERM_COUNT]=
{
    {
        {100013989.0,0,0},
        {1670700.0,3.0984635,6283.07585},
        {13956.0,3.05525,12566.1517},
        {3084.0,5.1985,77713.7715},
        {1628.0,1.1739,5753.3849},
        {1576.0,2.8469,7860.4194},
        {925.0,5.453,11506.77},
        {542.0,4.564,3930.21},
        {472.0,3.661,5884.927},
        {346.0,0.964,5507.553},
        {329.0,5.9,5223.694},
        {307.0,0.299,5573.143},
        {243.0,4.273,11790.629},
        {212.0,5.847,1577.344},
        {186.0,5.022,10977.079},
        {175.0,3.012,18849.228},
        {110.0,5.055,5486.778},
        {98,0.89,6069.78},
        {86,5.69,15720.84},
```

```
        {86,1.27,161000.69},
        {65,0.27,17260.15},
        {63,0.92,529.69},
        {57,2.01,83996.85},
        {56,5.24,71430.7},
        {49,3.25,2544.31},
        {47,2.58,775.52},
        {45,5.54,9437.76},
        {43,6.01,6275.96},
        {39,5.36,4694},
        {38,2.39,8827.39},
        {37,0.83,19651.05},
        {37,4.9,12139.55},
        {36,1.67,12036.46},
        {35,1.84,2942.46},
        {33,0.24,7084.9},
        {32,0.18,5088.63},
        {32,1.78,398.15},
        {28,1.21,6286.6},
        {28,1.9,6279.55},
        {26,4.59,10447.39}
    },
    {
        {103019.0,1.10749,6283.07585},
        {1721.0,1.0644,12566.1517},
```

```
        {702.0,3.142,0},
        {32,1.02,18849.23},
        {31,2.84,5507.55},
        {25,1.32,5223.69},
        {18,1.42,1577.34},
        {10,5.91,10977.08},
        {9,1.42,6275.96},
        {9,0.27,5486.78}
    },
    {
        {4359.0,5.7846,6283.0758},
        {124.0,5.579,12566.152},
        {12,3.14,0},
        {9,3.63,77713.77},
        {6,1.87,5573.14},
        {3,5.47,18849.23}
    },
    {
        {145.0,4.273,6283.076},
        {7,3.92,12566.15}
    },
    {
        {4,2.56,6283.08}
    }
};
```

```c

////////////////////////////////////////////////////////////////
///  Periodic Terms for the nutation in longitude and obliquity
////////////////////////////////////////////////////////////////

const int Y_TERMS[Y_COUNT][TERM_Y_COUNT]=
{
	{0,0,0,0,1},
	{-2,0,0,2,2},
	{0,0,0,2,2},
	{0,0,0,0,2},
	{0,1,0,0,0},
	{0,0,1,0,0},
	{-2,1,0,2,2},
	{0,0,0,2,1},
	{0,0,1,2,2},
	{-2,-1,0,2,2},
	{-2,0,1,0,0},
	{-2,0,0,2,1},
	{0,0,-1,2,2},
```

```
{2,0,0,0,0},
{0,0,1,0,1},
{2,0,-1,2,2},
{0,0,-1,0,1},
{0,0,1,2,1},
{-2,0,2,0,0},
{0,0,-2,2,1},
{2,0,0,2,2},
{0,0,2,2,2},
{0,0,2,0,0},
{-2,0,1,2,2},
{0,0,0,2,0},
{-2,0,0,2,0},
{0,0,-1,2,1},
{0,2,0,0,0},
{2,0,-1,0,1},
{-2,2,0,2,2},
{0,1,0,0,1},
{-2,0,1,0,1},
{0,-1,0,0,1},
{0,0,2,-2,0},
{2,0,-1,2,1},
{2,0,1,2,2},
{0,1,0,2,2},
{-2,1,1,0,0},
```

```
    {0,-1,0,2,2},
    {2,0,0,2,1},
    {2,0,1,0,0},
    {-2,0,2,2,2},
    {-2,0,1,2,1},
    {2,0,-2,0,1},
    {2,0,0,0,1},
    {0,-1,1,0,0},
    {-2,-1,0,2,1},
    {-2,0,0,0,1},
    {0,0,2,2,1},
    {-2,0,2,0,1},
    {-2,1,0,2,1},
    {0,0,1,-2,0},
    {-1,0,1,0,0},
    {-2,1,0,0,0},
    {1,0,0,0,0},
    {0,0,1,2,0},
    {0,0,-2,2,2},
    {-1,-1,1,0,0},
    {0,1,1,0,0},
    {0,-1,1,2,2},
    {2,-1,-1,2,2},
    {0,0,3,2,2},
    {2,-1,0,2,2},
```

```
};

const double PE_TERMS[Y_COUNT][TERM_PE_COUNT]={
    {-171996,-174.2,92025,8.9},
    {-13187,-1.6,5736,-3.1},
    {-2274,-0.2,977,-0.5},
    {2062,0.2,-895,0.5},
    {1426,-3.4,54,-0.1},
    {712,0.1,-7,0},
    {-517,1.2,224,-0.6},
    {-386,-0.4,200,0},
    {-301,0,129,-0.1},
    {217,-0.5,-95,0.3},
    {-158,0,0,0},
    {129,0.1,-70,0},
    {123,0,-53,0},
    {63,0,0,0},
    {63,0.1,-33,0},
    {-59,0,26,0},
    {-58,-0.1,32,0},
    {-51,0,27,0},
    {48,0,0,0},
    {46,0,-24,0},
    {-38,0,16,0},
    {-31,0,13,0},
```

```
{29,0,0,0},
{29,0,-12,0},
{26,0,0,0},
{-22,0,0,0},
{21,0,-10,0},
{17,-0.1,0,0},
{16,0,-8,0},
{-16,0.1,7,0},
{-15,0,9,0},
{-13,0,7,0},
{-12,0,6,0},
{11,0,0,0},
{-10,0,5,0},
{-8,0,3,0},
{7,0,-3,0},
{-7,0,0,0},
{-7,0,3,0},
{-7,0,3,0},
{6,0,0,0},
{6,0,-3,0},
{6,0,-3,0},
{-6,0,3,0},
{-6,0,3,0},
{5,0,0,0},
{-5,0,3,0},
```

```
    {-5,0,3,0},
    {-5,0,3,0},
    {4,0,0,0},
    {4,0,0,0},
    {4,0,0,0},
    {-4,0,0,0},
    {-4,0,0,0},
    {-4,0,0,0},
    {3,0,0,0},
    {-3,0,0,0},
    {-3,0,0,0},
    {-3,0,0,0},
    {-3,0,0,0},
    {-3,0,0,0},
    {-3,0,0,0},
    {-3,0,0,0},
};

///////////////////////////////////////////////////

double rad2deg(double radians)
{
    return (180.0/PI)*radians;
}

```

```
double deg2rad(double degrees)
{
    return (PI/180.0)*degrees;
}

int integer(double value)
{
  return value;
}

double limit_degrees(double degrees)
{
    double limited;

    degrees /= 360.0;
    limited = 360.0*(degrees-floor(degrees));
    if (limited < 0) limited += 360.0;

    return limited;
}

double limit_degrees180pm(double degrees)
{
    double limited;

```

```
    degrees /= 360.0;
    limited = 360.0*(degrees-floor(degrees));
    if      (limited < -180.0) limited += 360.0;
    else if (limited >  180.0) limited -= 360.0;

    return limited;
}

double limit_degrees180(double degrees)
{
    double limited;

    degrees /= 180.0;
    limited = 180.0*(degrees-floor(degrees));
    if (limited < 0) limited += 180.0;

    return limited;
}

double limit_zero2one(double value)
{
    double limited;

    limited = value - floor(value);
    if (limited < 0) limited += 1.0;
```

```
    return limited;
}

double limit_minutes(double minutes)
{
    double limited=minutes;

    if      (limited < -20.0) limited += 1440.0;
    else if (limited >  20.0) limited -= 1440.0;

    return limited;
}

double dayfrac_to_local_hr(double dayfrac, double timezone)
{
    return 24.0*limit_zero2one(dayfrac + timezone/24.0);
}

double third_order_polynomial(double a, double b, double c,
   double d, double x)
{
    return ((a*x + b)*x + c)*x + d;
}
```

```
// ////////////////////////////////////////////////////////////////////////
int validate_inputs(spa_data *spa)
{
    if ((spa->year        < -2000) || (spa->year        > 6000)) return 1;
    if ((spa->month       < 1    ) || (spa->month       > 12  )) return 2;
    if ((spa->day         < 1    ) || (spa->day         > 31  )) return 3;
    if ((spa->hour        < 0    ) || (spa->hour        > 24  )) return 4;
    if ((spa->minute      < 0    ) || (spa->minute      > 59  )) return 5;
    if ((spa->second      < 0    ) || (spa->second      >=60  )) return 6;
    if ((spa->pressure    < 0    ) || (spa->pressure    > 5000)) return 12;
    if ((spa->temperature <= -273) || (spa->temperature > 6000)) return 13;
    if ((spa->delta_ut1   <= -1  ) || (spa->delta_ut1   >= 1  )) return 17;
  if ((spa->hour        == 24  ) && (spa->minute      > 0   )) return 5;
```

```
    if ((spa->hour         == 24   ) && (spa->second      > 0   )) return 6;

    if (fabs(spa->delta_t)       > 8000     ) return 7;
    if (fabs(spa->timezone)      > 18       ) return 8;
    if (fabs(spa->longitude)     > 180      ) return 9;
    if (fabs(spa->latitude)      > 90       ) return 10;
    if (fabs(spa->atmos_refract) > 5        ) return 16;
    if (     spa->elevation      < -6500000) return 11;

    if ((spa->function == SPA_ZA_INC) || (spa->function == SPA_ALL))
    {
        if (fabs(spa->slope)         > 360) return 14;
        if (fabs(spa->azm_rotation)  > 360) return 15;
    }

    return 0;
}
///////////////////////////////////////////////////////////////////////////
double julian_day (int year, int month, int day, int hour, int minute, double second, double dut1, double tz)
{
```

```
    double day_decimal, julian_day, a;

    day_decimal = day + (hour - tz + (minute + (second + dut1)
        /60.0)/60.0)/24.0;

    if (month < 3) {
        month += 12;
        year--;
    }

    julian_day = integer(365.25*(year+4716.0)) + integer
        (30.6001*(month+1)) + day_decimal - 1524.5;

    if (julian_day > 2299160.0) {
        a = integer(year/100);
        julian_day += (2 - a + integer(a/4));
    }

    return julian_day;
}

double julian_century(double jd)
{
    return (jd-2451545.0)/36525.0;
}
```

```

double julian_ephemeris_day(double jd, double delta_t)
{
    return jd+delta_t/86400.0;
}

double julian_ephemeris_century(double jde)
{
    return (jde - 2451545.0)/36525.0;
}

double julian_ephemeris_millennium(double jce)
{
    return (jce/10.0);
}

double earth_periodic_term_summation(const double terms[][
    TERM_COUNT], int count, double jme)
{
    int i;
    double sum=0;

    for (i = 0; i < count; i++)
        sum += terms[i][TERM_A]*cos(terms[i][TERM_B]+terms[i][
            TERM_C]*jme);
```

```

    return sum;
}

double earth_values(double term_sum[], int count, double jme)
{
    int i;
    double sum=0;

    for (i = 0; i < count; i++)
        sum += term_sum[i]*pow(jme, i);

    sum /= 1.0e8;

    return sum;
}

double earth_heliocentric_longitude(double jme)
{
    double sum[L_COUNT];
    int i;

    for (i = 0; i < L_COUNT; i++)
        sum[i] = earth_periodic_term_summation(L_TERMS[i],
            l_subcount[i], jme);
```

```
    return limit_degrees(rad2deg(earth_values(sum, L_COUNT,
        jme)));

}

double earth_heliocentric_latitude(double jme)
{
    double sum[B_COUNT];
    int i;

    for (i = 0; i < B_COUNT; i++)
        sum[i] = earth_periodic_term_summation(B_TERMS[i],
            b_subcount[i], jme);

    return rad2deg(earth_values(sum, B_COUNT, jme));

}

double earth_radius_vector(double jme)
{
    double sum[R_COUNT];
    int i;

    for (i = 0; i < R_COUNT; i++)
```

```
        sum[i] = earth_periodic_term_summation(R_TERMS[i], r_subcount[i], jme);

    return earth_values(sum, R_COUNT, jme);

}

double geocentric_longitude(double l)
{
    double theta = l + 180.0;

    if (theta >= 360.0) theta -= 360.0;

    return theta;
}

double geocentric_latitude(double b)
{
    return -b;
}

double mean_elongation_moon_sun(double jce)
{
    return third_order_polynomial(1.0/189474.0, -0.0019142, 445267.11148, 297.85036, jce);
```

```
}

double mean_anomaly_sun(double jce)
{
    return third_order_polynomial(-1.0/300000.0, -0.0001603,
        35999.05034, 357.52772, jce);
}

double mean_anomaly_moon(double jce)
{
    return third_order_polynomial(1.0/56250.0, 0.0086972,
        477198.867398, 134.96298, jce);
}

double argument_latitude_moon(double jce)
{
    return third_order_polynomial(1.0/327270.0, -0.0036825,
        483202.017538, 93.27191, jce);
}

double ascending_longitude_moon(double jce)
{
    return third_order_polynomial(1.0/450000.0, 0.0020708,
        -1934.136261, 125.04452, jce);
}
```

```
double xy_term_summation(int i, double x[TERM_X_COUNT])
{
    int j;
    double sum=0;

    for (j = 0; j < TERM_Y_COUNT; j++)
        sum += x[j]*Y_TERMS[i][j];

    return sum;
}

void nutation_longitude_and_obliquity(double jce, double x[TERM_X_COUNT], double *del_psi,
                                      double *del_epsilon)
{
    int i;
    double xy_term_sum, sum_psi=0, sum_epsilon=0;

    for (i = 0; i < Y_COUNT; i++) {
        xy_term_sum  = deg2rad(xy_term_summation(i, x));
        sum_psi     += (PE_TERMS[i][TERM_PSI_A] + jce*PE_TERMS[i][TERM_PSI_B])*sin(xy_term_sum);
        sum_epsilon += (PE_TERMS[i][TERM_EPS_C] + jce*PE_TERMS[i][TERM_EPS_D])*cos(xy_term_sum);
```

```
    }

    *del_psi     = sum_psi     / 36000000.0;
    *del_epsilon = sum_epsilon / 36000000.0;
}

double ecliptic_mean_obliquity(double jme)
{
    double u = jme/10.0;

    return 84381.448 + u*(-4680.93 + u*(-1.55 + u*(1999.25 + u
        *(-51.38 + u*(-249.67 +
                      u*(  -39.05 + u*( 7.12 + u*(  27.87 + u
                         *(  5.79 + u*2.45)))))))));
}

double ecliptic_true_obliquity(double delta_epsilon, double
    epsilon0)
{
    return delta_epsilon + epsilon0/3600.0;
}

double aberration_correction(double r)
{
    return -20.4898 / (3600.0*r);
```

```
}

double apparent_sun_longitude(double theta, double delta_psi, double delta_tau)
{
    return theta + delta_psi + delta_tau;
}

double greenwich_mean_sidereal_time (double jd, double jc)
{
    return limit_degrees(280.46061837 + 360.98564736629 * (jd - 2451545.0) +
                                       jc*jc*(0.000387933 - jc/38710000.0));
}

double greenwich_sidereal_time (double nu0, double delta_psi, double epsilon)
{
    return nu0 + delta_psi*cos(deg2rad(epsilon));
}

double geocentric_right_ascension(double lamda, double epsilon, double beta)
{
```

```
    double lamda_rad   = deg2rad(lamda);
    double epsilon_rad = deg2rad(epsilon);

    return limit_degrees(rad2deg(atan2(sin(lamda_rad)*cos(
        epsilon_rad) -
                    tan(deg2rad(beta))*sin(epsilon_rad), cos(
                        lamda_rad))));
}

double geocentric_declination(double beta, double epsilon,
    double lamda)
{
    double beta_rad    = deg2rad(beta);
    double epsilon_rad = deg2rad(epsilon);

    return rad2deg(asin(sin(beta_rad)*cos(epsilon_rad) +
                        cos(beta_rad)*sin(epsilon_rad)*sin(
                            deg2rad(lamda))));
}

double observer_hour_angle(double nu, double longitude, double
    alpha_deg)
{
    return limit_degrees(nu + longitude - alpha_deg);
}
```

```

double sun_equatorial_horizontal_parallax(double r)
{
    return 8.794 / (3600.0 * r);
}

void right_ascension_parallax_and_topocentric_dec(double latitude, double elevation,
           double xi, double h, double delta, double *delta_alpha, double *delta_prime)
{
    double delta_alpha_rad;
    double lat_rad   = deg2rad(latitude);
    double xi_rad    = deg2rad(xi);
    double h_rad     = deg2rad(h);
    double delta_rad = deg2rad(delta);
    double u = atan(0.99664719 * tan(lat_rad));
    double y = 0.99664719 * sin(u) + elevation*sin(lat_rad)/6378140.0;
    double x =              cos(u) + elevation*cos(lat_rad)/6378140.0;

    delta_alpha_rad =       atan2(                - x*sin(xi_rad) *sin(h_rad),
                      cos(delta_rad) - x*sin(xi_rad) *cos(h_rad));
```

```

    *delta_prime = rad2deg(atan2((sin(delta_rad) - y*sin(xi_rad))*cos(delta_alpha_rad),
                    cos(delta_rad) - x*sin(xi_rad) *cos(h_rad)));

    *delta_alpha = rad2deg(delta_alpha_rad);
}

double topocentric_right_ascension(double alpha_deg, double delta_alpha)
{
    return alpha_deg + delta_alpha;
}

double topocentric_local_hour_angle(double h, double delta_alpha)
{
    return h - delta_alpha;
}

double topocentric_elevation_angle(double latitude, double delta_prime, double h_prime)
{
    double lat_rad         = deg2rad(latitude);
    double delta_prime_rad = deg2rad(delta_prime);
```

```

    return rad2deg(asin(sin(lat_rad)*sin(delta_prime_rad) +
                        cos(lat_rad)*cos(delta_prime_rad) * cos(deg2rad(h_prime))));
}

double atmospheric_refraction_correction(double pressure, double temperature,
                      double atmos_refract, double e0)
{
    double del_e = 0;

    if (e0 >= -1*(SUN_RADIUS + atmos_refract))
        del_e = (pressure / 1010.0) * (283.0 / (273.0 + temperature)) *
                         1.02 / (60.0 * tan(deg2rad(e0 + 10.3/(e0 + 5.11))));

    return del_e;
}

double topocentric_elevation_angle_corrected(double e0, double delta_e)
{
    return e0 + delta_e;
```

```
}

double topocentric_zenith_angle(double e)
{
    return 90.0 - e;
}

double topocentric_azimuth_angle_astro(double h_prime, double
    latitude, double delta_prime)
{
    double h_prime_rad = deg2rad(h_prime);
    double lat_rad     = deg2rad(latitude);

    return limit_degrees(rad2deg(atan2(sin(h_prime_rad),
                   cos(h_prime_rad)*sin(lat_rad) - tan(deg2rad(
                       delta_prime))*cos(lat_rad))));
}

double topocentric_azimuth_angle(double azimuth_astro)
{
    return limit_degrees(azimuth_astro + 180.0);
}

double surface_incidence_angle(double zenith, double
    azimuth_astro, double azm_rotation,
```

```
                    double slope)
{
    double zenith_rad = deg2rad(zenith);
    double slope_rad  = deg2rad(slope);

    return rad2deg(acos(cos(zenith_rad)*cos(slope_rad)  +
                        sin(slope_rad )*sin(zenith_rad) * cos(deg2rad(azimuth_astro - azm_rotation))));
}

double sun_mean_longitude(double jme)
{
    return limit_degrees(280.4664567 + jme*(360007.6982779 + jme*(0.03032028 +
                    jme*(1/49931.0   + jme*(-1/15300.0     + jme*(-1/2000000.0))))));
}

double eot(double m, double alpha, double del_psi, double epsilon)
{
    return limit_minutes(4.0*(m - 0.0057183 - alpha + del_psi*cos(deg2rad(epsilon))));
}
```

```
double approx_sun_transit_time(double alpha_zero, double
    longitude, double nu)
{
    return (alpha_zero - longitude - nu) / 360.0;
}

double sun_hour_angle_at_rise_set(double latitude, double
    delta_zero, double h0_prime)
{
    double h0              = -99999;
    double latitude_rad    = deg2rad(latitude);
    double delta_zero_rad  = deg2rad(delta_zero);
    double argument        = (sin(deg2rad(h0_prime)) - sin(
        latitude_rad)*sin(delta_zero_rad)) /
                             (cos(latitude_rad)*cos(delta_zero_rad));

    if (fabs(argument) <= 1) h0 = limit_degrees180(rad2deg(
        acos(argument)));

    return h0;
}

void approx_sun_rise_and_set(double *m_rts, double h0)
{
```

```
    double h0_dfrac = h0/360.0;

    m_rts[SUN_RISE]     = limit_zero2one(m_rts[SUN_TRANSIT] -
        h0_dfrac);
    m_rts[SUN_SET]      = limit_zero2one(m_rts[SUN_TRANSIT] +
        h0_dfrac);
    m_rts[SUN_TRANSIT] = limit_zero2one(m_rts[SUN_TRANSIT]);
}

double rts_alpha_delta_prime(double *ad, double n)
{
    double a = ad[JD_ZERO] - ad[JD_MINUS];
    double b = ad[JD_PLUS] - ad[JD_ZERO];

    if (fabs(a) >= 2.0) a = limit_zero2one(a);
    if (fabs(b) >= 2.0) b = limit_zero2one(b);

    return ad[JD_ZERO] + n * (a + b + (b-a)*n)/2.0;
}

double rts_sun_altitude(double latitude, double delta_prime,
    double h_prime)
{
    double latitude_rad    = deg2rad(latitude);
    double delta_prime_rad = deg2rad(delta_prime);
```

```c

    return rad2deg(asin(sin(latitude_rad)*sin(delta_prime_rad) +
                        cos(latitude_rad)*cos(delta_prime_rad)*cos(deg2rad(h_prime))));
}

double sun_rise_and_set(double *m_rts,   double *h_rts,   double *delta_prime, double latitude,
                        double *h_prime, double h0_prime, int sun)
{
    return m_rts[sun] + (h_rts[sun] - h0_prime) /
          (360.0*cos(deg2rad(delta_prime[sun]))*cos(deg2rad(latitude))*sin(deg2rad(h_prime[sun])));
}

/////////////////////////////////////////////////////////////////////////////////////////////
// Calculate required SPA parameters to get the right ascension (alpha) and declination (delta)
// Note: JD must be already calculated and in structure
/////////////////////////////////////////////////////////////////////////////////////////////
```

```c
void calculate_geocentric_sun_right_ascension_and_declination(
    spa_data *spa)
{
    double x[TERM_X_COUNT];

    spa->jc = julian_century(spa->jd);

    spa->jde = julian_ephemeris_day(spa->jd, spa->delta_t);
    spa->jce = julian_ephemeris_century(spa->jde);
    spa->jme = julian_ephemeris_millennium(spa->jce);

    spa->l = earth_heliocentric_longitude(spa->jme);
    spa->b = earth_heliocentric_latitude(spa->jme);
    spa->r = earth_radius_vector(spa->jme);

    spa->theta = geocentric_longitude(spa->l);
    spa->beta  = geocentric_latitude(spa->b);

    x[TERM_X0] = spa->x0 = mean_elongation_moon_sun(spa->jce);
    x[TERM_X1] = spa->x1 = mean_anomaly_sun(spa->jce);
    x[TERM_X2] = spa->x2 = mean_anomaly_moon(spa->jce);
    x[TERM_X3] = spa->x3 = argument_latitude_moon(spa->jce);
    x[TERM_X4] = spa->x4 = ascending_longitude_moon(spa->jce);

```

```
    nutation_longitude_and_obliquity(spa->jce, x, &(spa->del_psi), &(spa->del_epsilon));

    spa->epsilon0 = ecliptic_mean_obliquity(spa->jme);
    spa->epsilon  = ecliptic_true_obliquity(spa->del_epsilon, spa->epsilon0);

    spa->del_tau   = aberration_correction(spa->r);
    spa->lamda     = apparent_sun_longitude(spa->theta, spa->del_psi, spa->del_tau);
    spa->nu0       = greenwich_mean_sidereal_time (spa->jd, spa->jc);
    spa->nu        = greenwich_sidereal_time (spa->nu0, spa->del_psi, spa->epsilon);

    spa->alpha = geocentric_right_ascension(spa->lamda, spa->epsilon, spa->beta);
    spa->delta = geocentric_declination(spa->beta, spa->epsilon, spa->lamda);
}

////////////////////////////////////////////////////////////////////////
```

```c
// Calculate Equation of Time (EOT) and Sun Rise, Transit, & Set (RTS)
// ////////////////////////////////////////////////////////////////////////

void calculate_eot_and_sun_rise_transit_set(spa_data *spa)
{
    spa_data sun_rts;
    double nu, m, h0, n;
    double alpha[JD_COUNT], delta[JD_COUNT];
    double m_rts[SUN_COUNT], nu_rts[SUN_COUNT], h_rts[SUN_COUNT];
    double alpha_prime[SUN_COUNT], delta_prime[SUN_COUNT], h_prime[SUN_COUNT];
    double h0_prime = -1*(SUN_RADIUS + spa->atmos_refract);
    int i;

  sun_rts  = *spa;
    m        = sun_mean_longitude(spa->jme);
    spa->eot = eot(m, spa->alpha, spa->del_psi, spa->epsilon);

    sun_rts.hour = sun_rts.minute = sun_rts.second = 0;
  sun_rts.delta_ut1 = sun_rts.timezone = 0.0;

```

```
    sun_rts.jd = julian_day (sun_rts.year,   sun_rts.month,  sun_rts.day,      sun_rts.hour,
                             sun_rts.minute, sun_rts.second, sun_rts.delta_ut1, sun_rts.timezone);

    calculate_geocentric_sun_right_ascension_and_declination(&sun_rts);
    nu = sun_rts.nu;

    sun_rts.delta_t = 0;
    sun_rts.jd--;
    for (i = 0; i < JD_COUNT; i++) {
        calculate_geocentric_sun_right_ascension_and_declination(&sun_rts);
        alpha[i] = sun_rts.alpha;
        delta[i] = sun_rts.delta;
        sun_rts.jd++;
    }

    m_rts[SUN_TRANSIT] = approx_sun_transit_time(alpha[JD_ZERO], spa->longitude, nu);
    h0 = sun_hour_angle_at_rise_set(spa->latitude, delta[JD_ZERO], h0_prime);

```

```
    if (h0 >= 0) {

        approx_sun_rise_and_set(m_rts, h0);

        for (i = 0; i < SUN_COUNT; i++) {

            nu_rts[i]      = nu + 360.985647*m_rts[i];

            n              = m_rts[i] + spa->delta_t/86400.0;
            alpha_prime[i] = rts_alpha_delta_prime(alpha, n);
            delta_prime[i] = rts_alpha_delta_prime(delta, n);

            h_prime[i]     = limit_degrees180pm(nu_rts[i] +
                spa->longitude - alpha_prime[i]);

            h_rts[i]       = rts_sun_altitude(spa->latitude,
                delta_prime[i], h_prime[i]);
        }

        spa->srha = h_prime[SUN_RISE];
        spa->ssha = h_prime[SUN_SET];
        spa->sta  = h_rts[SUN_TRANSIT];

        spa->suntransit = dayfrac_to_local_hr(m_rts[
            SUN_TRANSIT] - h_prime[SUN_TRANSIT] / 360.0,
```

```
                                                    spa->timezone);

        spa->sunrise = dayfrac_to_local_hr(sun_rise_and_set(
            m_rts, h_rts, delta_prime,
                                spa->latitude, h_prime, h0_prime,
                                    SUN_RISE), spa->timezone);

        spa->sunset  = dayfrac_to_local_hr(sun_rise_and_set(
            m_rts, h_rts, delta_prime,
                                spa->latitude, h_prime, h0_prime,
                                    SUN_SET),  spa->timezone);

    } else spa->srha= spa->ssha= spa->sta= spa->suntransit=
        spa->sunrise= spa->sunset= -99999;

}

//
    ////////////////////////////////////////////////////////////////////////////

// Calculate all SPA parameters and put into structure
// Note: All inputs values (listed in header file) must
    already be in structure
//
    ////////////////////////////////////////////////////////////////////////////
```

```
int spa_calculate(spa_data *spa)
{
    int result;

    result = validate_inputs(spa);

    if (result == 0)
    {
        spa->jd = julian_day (spa->year,    spa->month,   spa->
            day,         spa->hour,
                              spa->minute, spa->second, spa->
                                  delta_ut1, spa->timezone);

        calculate_geocentric_sun_right_ascension_and_declination
            (spa);

        spa->h  = observer_hour_angle(spa->nu, spa->longitude,
             spa->alpha);
        spa->xi = sun_equatorial_horizontal_parallax(spa->r);

        right_ascension_parallax_and_topocentric_dec(spa->
            latitude, spa->elevation, spa->xi,
                                      spa->h, spa->delta, &(spa->
                                          del_alpha), &(spa->
```

```
delta_prime));

    spa->alpha_prime = topocentric_right_ascension(spa->alpha, spa->del_alpha);
    spa->h_prime     = topocentric_local_hour_angle(spa->h, spa->del_alpha);

    spa->e0      = topocentric_elevation_angle(spa->latitude, spa->delta_prime, spa->h_prime);
    spa->del_e   = atmospheric_refraction_correction(spa->pressure, spa->temperature,
                                                     spa->atmos_refract, spa->e0);
    spa->e       = topocentric_elevation_angle_corrected(spa->e0, spa->del_e);

    spa->zenith        = topocentric_zenith_angle(spa->e);
    spa->azimuth_astro = topocentric_azimuth_angle_astro(spa->h_prime, spa->latitude,
```

```
            spa->azimuth         = topocentric_azimuth_angle(spa->azimuth_astro);

            if ((spa->function == SPA_ZA_INC) || (spa->function == SPA_ALL))
                spa->incidence  = surface_incidence_angle(spa->zenith, spa->azimuth_astro,
                                                          spa->azm_rotation, spa->slope);

            if ((spa->function == SPA_ZA_RTS) || (spa->function == SPA_ALL))
                calculate_eot_and_sun_rise_transit_set(spa);
        }

    return result;
}
/////////////////////////////////////////////////////////////////////////////////
```

Listing I.6: SPA Source File