CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

MAZE SOLVING ROBOT USING A* ALGORITHM

A graduate project submitted in partial fulfillment of requirements

For the degree of Master of Science in Computer Engineering

By

Vedant Ujwal Vartak

August 2023

The graduate project of Vedant Ujwal Vartak is approved:

Prof. Saba Janamian	Date
Dr. Md Sahabul Alam	Date
Dr. Shahnam Mirzaei, Chair	Date

California State University, Northridge

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my Project Chair, Dr. Shahnam Mirzaei, for arranging this project idea and allowing me to work on it. I would like to thank him for guiding me throughout the project and giving me valuable insights.

I would also like to thank Prof. Saba Janamian and Dr. Md Sahabul Alam for their contribution to the completion of this project as well as the Head of Department Dr. Xiaojun (Ashley) Geng for guiding and advising me with my academics throughout my degree.

Special thanks to my friend Avinash Damse who helped me become familiar with the software and programming environment needed for this project as well as helping me in the implementation process.

Lastly, I am forever indebted to my parents who have always supported me with everything as well as all my friends, especially Aditee and Akshata for their encouragement and support.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Maze Solving Robot using A* algorithm

By

Vedant Ujwal Vartak

Master of Science in Computer Engineering

Maze solving is the process of figuring out how to move from one place to another through a maze. This topic has been a contentious study topic in the disciplines of robotics and computer science for many years. The objective of solving a maze is to locate the shortest or most efficient path to the destination while avoiding obstacles and dead ends.

AI can be used to solve mazes by figuring out the best path to travel to get to the desired location by training a machine learning model to recognize the various patterns and structures present in the maze. AI maze-solving techniques include using a deep reinforcement learning algorithm. In this project, the robot learns to navigate the maze by receiving feedback based on its behavior. After receiving the feedback, when the robot is asked to navigate the maze the second time, it will use the previous feedback which would help to cut down on unnecessary routes and lead the robot onto the shortest path. This will help to navigate the maze in the shortest possible time.

The algorithm used in this project is the A* algorithm and the development board which is used is the MSP432P401R board by Texas Instruments which is programmed using the Code Composer Studio IDE.

CHAPTER 1 : INTRODUCTION

1.1 Background

A maze is a complicated system of pathways or alleys that is usually created as a task or puzzle for a person or animal to travel through. Although mazes occur in a wide variety of sizes and designs, they all have the same basic components: a starting point, one or more ending locations, and a network of connected paths or walls that lead to dead ends and demand the navigator to make judgments.

The usage of mazes as religious and symbolic structures dates back thousands of years to the prehistoric era. Mazes gained popularity as garden centerpieces and were frequently used as spaces for introspection and meditation in medieval Europe.

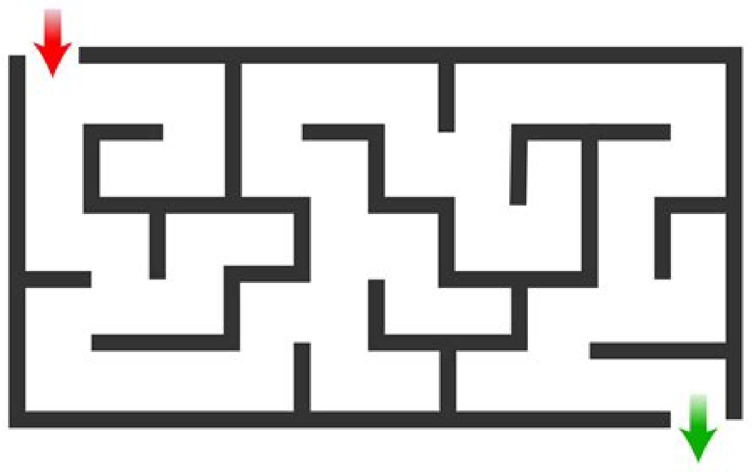


Figure 1.1: A simple maze

There are various kinds of mazes, such as:

- Traditional maze designs with branching paths and dead ends are known as classic mazes.
- Labyrinths which are unicursal mazes, which means there is only one path that leads continuously inside and then outward to the center.
- Network mazes having numerous entries and exits and complicated pathways that cross over and link.

- Virtual mazes which can be navigated on a computer screen or in a virtual reality environment. These mazes are made using computer software.

Bushes, wood, stone, or paper are just a few examples of the materials that can be used to make mazes. Even cornfields and other organic materials have been used to construct some mazes.

It takes perseverance, planning, and problem-solving abilities to get through a maze. This led to the idea of maze-solving as a source of recreation. People of all ages can appreciate it as a demanding and entertaining activity. The game of maze solving has been popular for a long time.

The concept of maze dates to around 10th century and is believed to have originated in Egypt where they used it as a form of entertainment and mystery which later became more popular in the Renaissance era in the form of maze gardens and hedges.

Figure 1.2: Maze gardens in ancient times

Today, mazes are incorporated for a wide range of activities, including teaching, entertainment, and research. They are utilized to study animal behavior and can be found in theme parks, computer games, literature, and even amusement parks.

The idea of maze solving brought forward many different strategies and ideas on how to solve the maze. The "left-hand rule," which implies placing your left hand on the maze's left wall and following it until you reach the exit, is a popular method for navigating mazes. This strategy works for mazes with a single entrance, a single exit, and no dead ends or loops. The "right-hand rule," which is identical to the "left-hand rule" but involves placing your right hand on the right wall of the maze instead, is another often used tactic. Other approaches, including drawing a map of the maze on paper with a pencil or utilizing a computer program to determine the shortest way, could be required for more complicated mazes.

Moreover, a wide variety of algorithms have been created expressly for resolving mazes, such as the flood-fill algorithm, which includes marking each maze cell that has been traversed and systematically investigating all adjacent cells until the exit is located.

In addition to being entertaining, maze solving has useful applications in industries like robotics and artificial intelligence. Several researchers train robots to navigate through actual situations, such as factories or hospitals, by using maze-solving algorithms.

1.2 Maze Solving using Algorithms

Use of artificial intelligence for maze solving made way for various algorithms which make it easy for solving complex problems. Algorithms for solving mazes are those that can locate a route from one end of a maze to the other. Several disciplines, including robotics, computer science, and game creation, utilize these algorithms.

Maze-solving algorithms come in a variety of forms, each with distinct advantages and disadvantages. The most typical varieties include:

- Wall Follower Algorithm: It entails following a maze wall, typically the left or right wall, until the exit is discovered. This approach can fail in more difficult mazes that have loops and backtracking, but it excels in straightforward ones.
- Recursive Backtracking Algorithm: This algorithm searches all potential paths until it discovers the exit. The algorithm goes back to the previous junction and tries an alternative path if a dead end is reached. For mazes with several solutions, this strategy works well, but it can be slow for bigger mazes.

- Breadth First Search Algorithm: Using the breadth-first search technique, all potential routes are investigated concurrently, beginning at the starting point and moving outward in all directions until the exit is located. The shortest way across a maze can be found using this method, however, it can be memory-intensive for bigger mazes.
- Depth First Search Algorithm: Like the recursive backtracking technique, the depth-first search algorithm fully analyzes each path before retracing. For bigger mazes, this approach may be quicker than the breadth-first search strategy.
- A* Algorithm: The A* algorithm prioritizes routes that are more likely to go to the exit using heuristics. Although it uses more computing power than other methods, this technique is particularly effective for locating the shortest route around a maze.

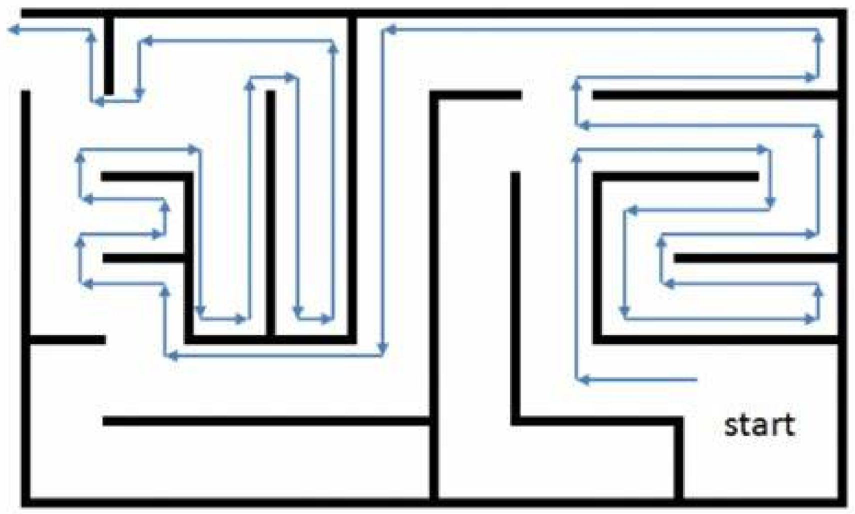


Figure 1.3: A visual of how the wall following algorithm functions

Maze-solving algorithms can be utilized in a variety of applications, including games, navigation, and robotics, and they can be implemented in several different programming languages. The ideal outcome, size, and difficulty of the maze are all important considerations for selecting the best algorithm.

1.3 Robotics

Robots are electromechanical machines that can carry out tasks that are typically done by humans or animals. Computer science, mechanical engineering, electrical engineering, and control systems are all combined in robotics. [12] Robots come in a wide variety of forms, from simple machines that can only do one thing to complicated machines that can do many things. Industrial robots, medical robots, service robots, and humanoid robots are a few examples of common robot kinds.

In production and assembly lines, industrial robots are employed to carry out repetitive operations like welding, painting, and material handling. To increase accuracy and reduce patient hazards, medical robots are employed in surgery and other medical treatments. There are several uses for service robots, including cleaning, security, and entertainment. Robots that resemble humans physically and are employed for research and development are called humanoid robots.

Ultimately, the topic of robotics is one that is fascinating and evolving quickly and has the potential to drastically alter many facets of our lives.

1.3.1 Use of Robotics in Artificial Intelligence

Artificial intelligence (AI) is a key component in enabling robots to carry out complicated tasks and interact with their environment, making robotics and AI closely related subjects. A wide range of technologies, such as machine learning, natural language processing, computer vision, and robotics, are included in the broad topic of artificial intelligence (AI).

Robotics heavily use machine learning, a crucial branch of artificial intelligence. Robots can learn from their mistakes and gradually increase their performance thanks to machine learning algorithms. Using machine learning algorithms to assess sensor data and make decisions based on that data, a robot, for instance, can learn to navigate its environment.

In general, AI is crucial in enabling robots to carry out difficult jobs and communicate with their surroundings in a more intelligent and natural way. We may anticipate seeing ever more sophisticated robotics applications that push the envelope of what is feasible with this fascinating and quickly expanding sector as AI technologies continue to grow.

1.4 Maze solving with Robots

With the help of algorithms, researchers came up with the idea that robots can be designed based on certain algorithms which will help to solve a practical maze rather than a virtual maze on a computer. Robots that use maze solving algorithms to go from one place in a maze to another are known as "maze solving robots." These robots are employed in many different fields, such as research, entertainment, and teaching.

Robots designed to solve mazes often utilize sensors to locate themselves within the maze and detect the boundaries of the maze. The best route to the exit is then determined using a maze-solving algorithm.

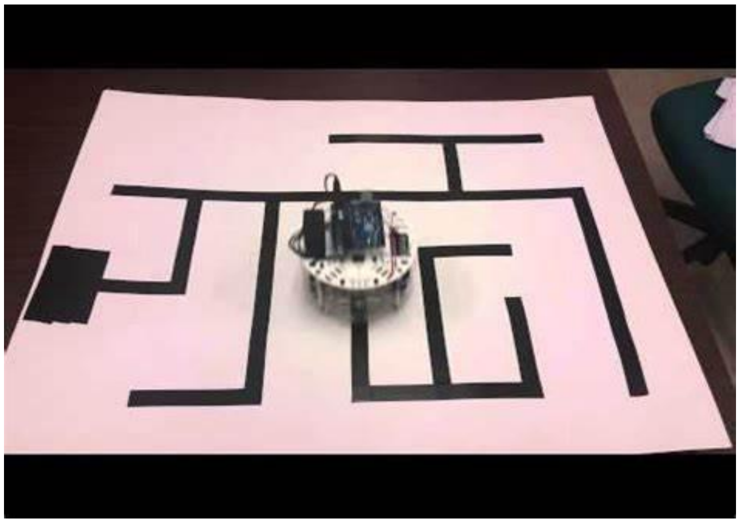


Figure 1.4: Line-following robot using line-following algorithm

Robots that can solve mazes come in a variety of designs, each with unique advantages and disadvantages. The most typical varieties include:

- Line-following robots: Robots that follow a line on the ground to move through a maze are known as line-following robots. Usually, they employ sensors to recognize the line and change their course accordingly.

- Wall-following robots: Robots that follow walls. These robots move through a maze by adhering to the walls. Usually, they employ sensors to recognize the walls and change their course accordingly.
- Autonomous robots: Robots that can move independently: These robots can navigate a maze using sophisticated sensors like cameras and lidar.

1.5 Report Organization

The report has been outlined in the following way. Chapter 1 is the introduction to the topic including an introduction to mazes, robotics and maze solving algorithms. Chapter 2 covers the project overview which explains what is done in this project along with objectives and workflow. Chapter 3 gives all details about the features and parameters of the MSP432P401R developmental board and its components which are used in this project and also gives background about integrated circuits and microcontrollers. Chapter 4 explains the Code Composer Studio IDE which is used to develop the program for this project. Chapter 5 includes an explanation about A* algorithm which is the main algorithm used in this project. It also explains how the algorithm works andng with its workflow. Chapter 6 tells how the project was implemented by integrating the hardware and software and making it to see the maze. Chapter 7 is the final chapter which is the results and conclusion section which gives the final summary about tofoject and also tells about future scope of the project.

CHAPTER 2 : PROJECT OVERVIEW

2.1 Project concept

This project on maze solving robot involves building a robot that can navigate through a maze and find its way to the end. The robot is designed to solve mazes autonomously, without any human intervention.

This project is built on the MSP432P401R development board by Texas Instruments which includes the microcontroller MSP432P401R launchpad, plus a variety of sensors, such as bump switches and proximity sensors. While walls and other barriers are detected by bump switches, the proximity of walls and other objects is determined by proximity sensors.

The robot is designed to begin at the maze's entrance and advance until it finds a wall or other obstruction. Following the wall, it navigates around corners and obstructions by using bump switches and proximity sensors. The robot is programmed to halt at the maze's conclusion and communicate that it has done so. The robot is programmed using an application called Code Composer Studio which is developed by Texas Instruments itself. The source code for the robot is developed in this application using C++ programming language and the robot is programmed with it. The source code for this project is developed using the A* algorithm which is used to find the shortest path between starthe t point and end point.

2.2 Project Objectives

- Understand how maze solving robots work by referring to previous research papers.
- Design a robot using MSP432P401R development board made by Texas Instruments
- Understand A* algorithm and figure out a way to use it to solve maze
- Develop a code for finding the shortest path to solve the maze using A* algorithm in with the help of Code Composer Studio
- Making a 6’ * 6’ maze to test the robot

2.3 Project flow

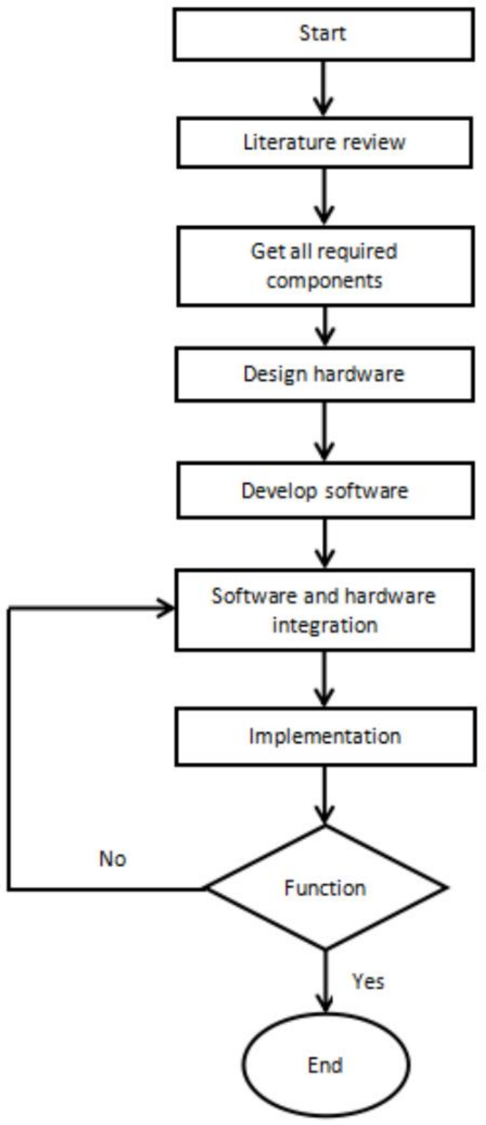


Figure 2.1: Workflow of the project

The important step is the function part where it is checked if the robot is functioning properly and giving the desired results. If this is not the case, then the software and hardware step is revisited and after tracing the issue and making necessary changes the next steps are followed again.

CHAPTER 3 : MSP432P401R DEVELOPMENT BOARD BY TEXAS INSTRUMENTS

3.1 Integrated Circuits

An integrated circuit (IC), also known as a microchip, is a miniature electronic circuit made up of numerous interconnected electronic parts including transistors, capacitors, and resistors that are assembled on a piece of semiconductor material, typically silicon. The development of integrated circuits, which allowed numerous electrical components to be combined onto a single chip to create smaller, lighter, and more effective electronic devices, completely transformed the electronics industry. Electronic gadgets such as computers, smartphones, televisions, cars, and medical equipment all make use of integrated circuits. They are also utilized in industrial applications like sensors and control systems. [17]

The two primary categories of integrated circuits are digital and analog. Modern digital electronics, including computers and smartphones, are built on digital integrated circuits, which are utilized to carry out logical processes. Contrarily, analog integrated circuits are utilized in a wide range of applications, including amplifiers and filters, to process continuous signals like sound or light.

3.2 Microcontrollers

A microcontroller is a little computer on a single integrated circuit. A central processing unit (CPU), memory, and a number of peripheral devices, including timers, serial ports, and analog-to-digital converters, make up this system. Microcontrollers are made to be integrated into other systems and devices in order to regulate their behavior. Microcontrollers are utilized in many different applications, such as robots, medical equipment, automotive systems, and home automation. They can be programmed in assembly language or high-level programming languages like C or C++. A few well-known producers of microcontrollers are Texas Instruments, Microchip Technologies, and Atmel.

3.3 Development board

A development board is a circuit board created to make the design and testing of electrical circuits simpler. A microcontroller or other integrated circuit is often found on these boards, along with any other components that are required for operation, such as power regulators, clock oscillators, and communication interfaces. Engineers, hobbyists, and students can prototype and test electronic circuits using development boards. They offer a simple platform for experimenting with various circuit designs and putting novel concepts to the test. Many development boards also come with pre-written software libraries and example code that may be used to have the microcontroller or other integrated circuit programmed right away.

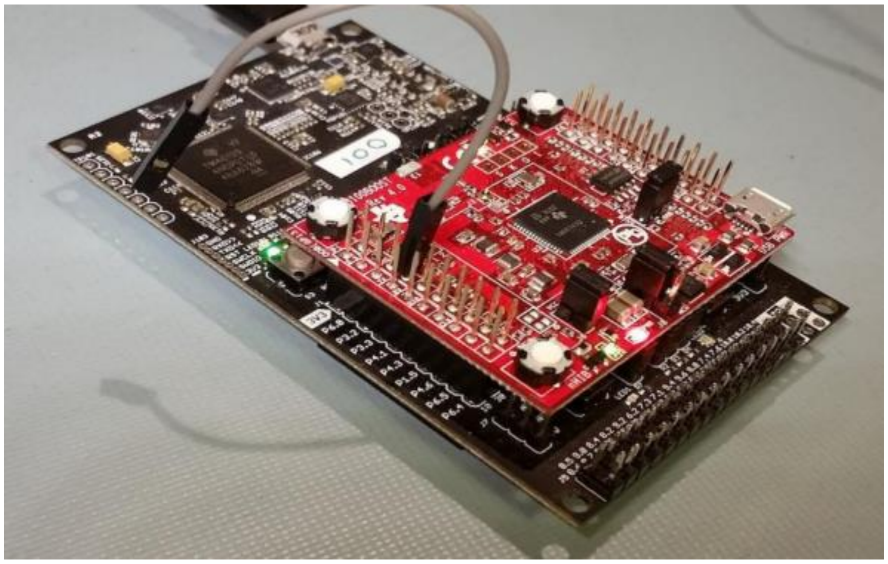


Figure 3.1: The MSP432P401R launchpad

3.4 MSP432P401R Development Board

Texas Instruments created the MSP432P401R Development Board, a microcontroller board that offers an affordable platform for creating and testing MSP432-based applications. It is a 32-bit ARM Cortex-M4F microcontroller with low power consumption and battery power capabilities. [1][3]. The MSP432 microcontroller, was an upgrade on the MSP430 microcontroller, with a number of advancements and modifications. Key improvements between MSP432 and MSP430 include:

- Architecture: The MSP430 has a 16-bit core while the MSP432 employs a 32-bit ARM Cortex-M4F core. The MSP432 is a more potent microcontroller thanks to its 32-bit design, which enables it to analyze more data simultaneously and carry out more difficult tasks.
- Clock speed: Compared to the MSP430, the MSP432 has a faster clock speed, with a maximum clock speed of 48 MHz as opposed to 25 MHz for the MSP430. This makes the MSP432 more effective by enabling faster instruction processing.
- Memory: The MSP432 offers up to 256 KB of flash memory and 64 KB of SRAM, which is more memory than the MSP430. The MSP432 can handle more complicated applications and store more data because to its expanded memory.
- Peripherals: In comparison to the MSP430, the MSP432 has more peripheral modules, including a real-time clock (RTC), a digital signal processor (DSP), and more sophisticated communication interfaces like USB and Ethernet.
- Power usage: The MSP432 uses less power than the MSP430 and has a number of low-power modes that can be employed to save energy. The MSP432 is hence perfect for applications that run on batteries.

Overall, compared to the MSP430 microprocessor, the MSP432 microcontroller has better performance, more features, and lower power consumption, making it a more potent and adaptable microcontroller for a variety of applications.

An MSP432P401R microcontroller is one of several features and peripherals on the development board that make it simple to create and test applications. These features allow programmers to create, test, and debug their applications. They also include an integrated USB port, on-board energy trace technology, and an onboard emulator/debugger. The board also features a range of input and output pins, including UART, SPI, and I2C interfaces, as well as digital and analog I/O. As a result, it is simple to link with other gadgets and sensors including motor drivers, LCD displays, and proximity sensors.

A strong software ecosystem like the Energia IDE offers a straightforward and user-friendly platform for programming the MSP432 microcontroller using the Wiring/Arduino programming language. It supports the development board's hardware features in addition to its hardware features. The MSP432P401R LaunchPad features an integrated debugger and supports JTAG-based programming and debugging. Moreover, it has a USB port for connecting to a host computer, and it may be powered by either a USB port or an external power source. For user contact and response, the LaunchPad board has two programmable buttons and lights. Additionally, it contains on-board energyTrace+ technology that allows programmers to monitor how much energy their applications use and reduce it.

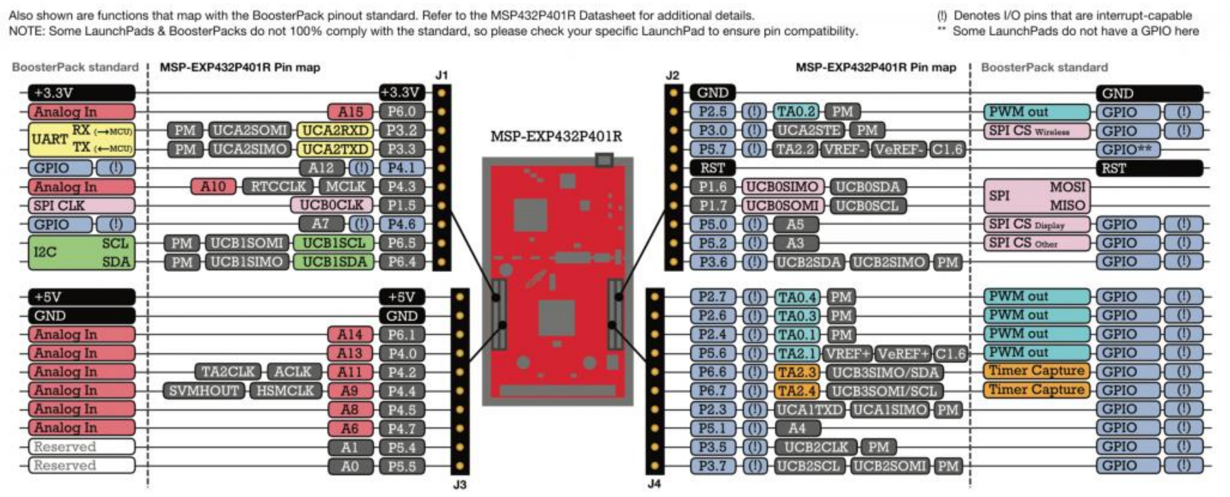

Also shown are functions that map with the BoosterPack pinout standard. Refer to the MSP432P401R Datasheet for additional details.
NOTE: Some LaunchPads & BoosterPacks do not 100% comply with the standard, so please check your specific LaunchPad to ensure pin compatibility.
(!) Denotes I/O pins that are interrupt-capable
** Some LaunchPads do not have a GPIO here
BoosterPack standard
MSP-EXP432P401R Pin map
MSP-EXP432P401R
J1
J2
J3
J4

Figure 3.2: Pin diagram of MSP432P401R [1]

3.4.1 Features

- Embedded microcontroller
  - 16-bit RISC architecture up to 16-MHz clock
  - Up to 256KB of ferroelectric random-access memory (FRAM)
    - Ultra-low-power writes
    - Fast write at 125 ns per word (64KB in 4 ms)
    - Flexible allocation of data and application code in memory
    - 1015 write cycle endurance
    - Radiation resistant and nonmagnetic

- Wide supply voltage ranges from 3.6 V down to 1.8 V (minimum supply voltage is restricted by SVS levels, see the SVS specifications)

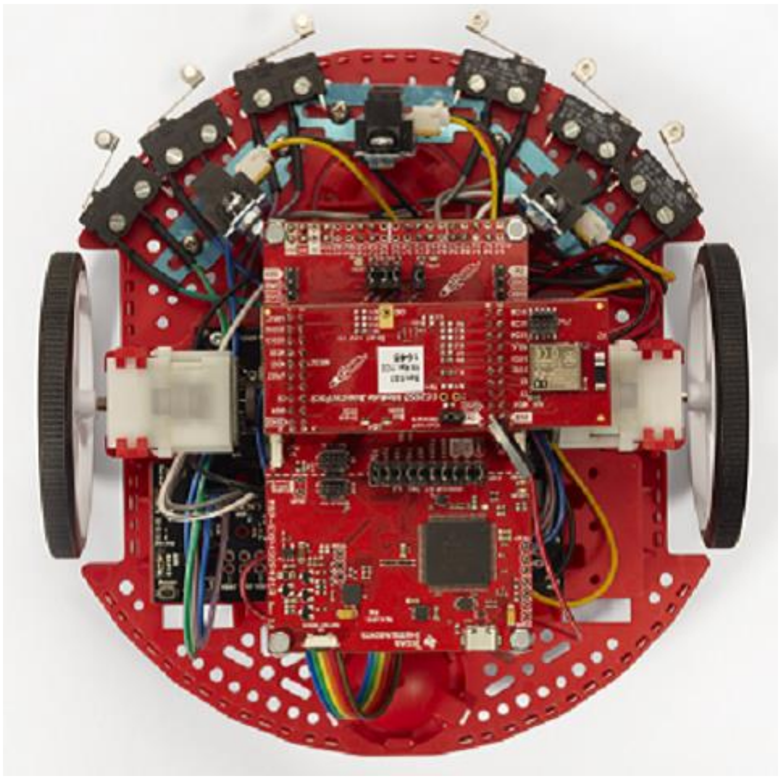


Figure 3.3: Texas Instrument’s RSLK robot built from the training kit

- Optimized ultra-low power modes
  - Active mode: 118 µA/MHz
  - Standby with VLO (LPM3): 500 nA
  - Standby with real-time clock (RTC) (LPM3.5): 350 nA (the RTC is clocked by a 3.7-pF crystal)
  - Shutdown (LPM4.5): 45 nA
- Low-energy accelerator (LEA) for signal processing (MSP430FR599x only)
  - Operation independent of CPU
  - 4KB of RAM shared with CPU

3.4.2 Parameters

Frequency (MHz)	16
Non-volatile memory (kB)	256
RAM (kB)	8
ADC type	12-bit SAR
Number of ADC channels	20
Number of GPIOs	68
Features	DMA, Low energy accelerator (LEA), real-time clock
UART	4
USB	No
Number of I2Cs	4
SPI	8
Number of comparator channels	16
Timers 16-bit	6
Bootloaders (BSL)	UART
Operating temperature range (°C)	-40 to 85

Table 1: Various parameters of MSP432 microcontroller

Overall, Texas Instruments' MSP432P401R Development Board is a strong and adaptable platform for creating and testing software based on the MSP432 microprocessor. It is a great option for anyone wishing to begin microcontroller programming and development because of its combination of hardware and software features.

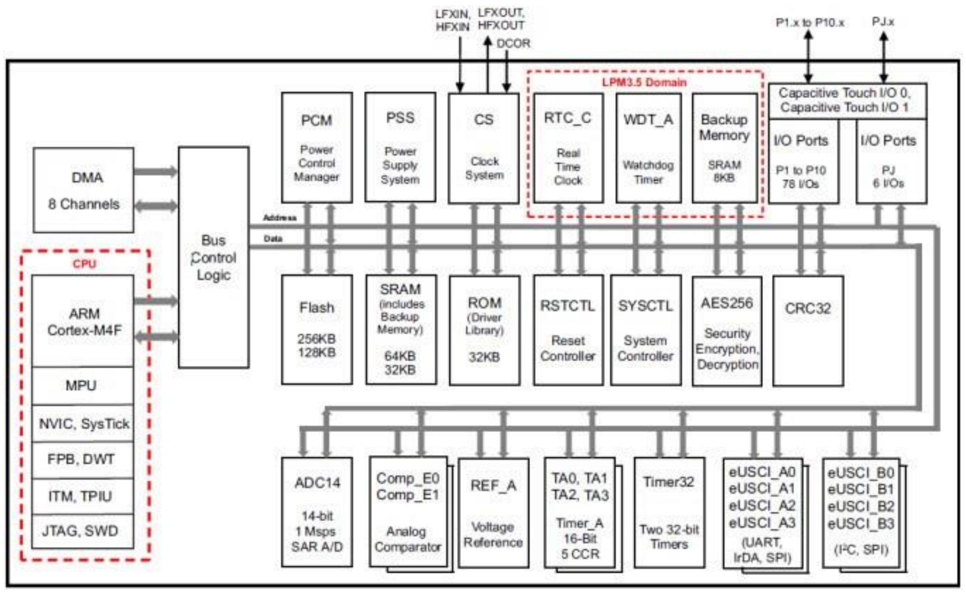

LFXIN, HFXIN
LFXOUT, HFXOUT
DCOR
P1.x to P10.x
PJ.x
LPM3.5 Domain
PCM Power Control Manager
PSS Power Supply System
CS Clock System
RTC_C Real Time Clock
WDT_A Watchdog Timer
Backup Memory SRAM 8KB
Capacitive Touch I/O 0, Capacitive Touch I/O 1
I/O Ports P1 to P10 78 I/Os
I/O Ports PJ 6 I/Os
DMA 8 Channels
Bus Control Logic
Address
Data
CPU
ARM Cortex-M4F
MPU
NVIC, SysTick
FPB, DWT
ITM, TPIU
JTAG, SWD
Flash 256KB 128KB
SRAM (includes Backup Memory) 64KB 32KB
ROM (Driver Library) 32KB
RSTCTL Reset Controller
SYSCTL System Controller
AES256 Security Encryption, Decryption
CRC32
ADC14 14-bit 1 Msps SAR A/D
Comp_E0 Comp_E1 Analog Comparator
REF_A Voltage Reference
TA0, TA1 TA2, TA3 Timer_A 16-Bit 5 CCR
Timer32 Two 32-bit Timers
eUSCI_A0 eUSCI_A1 eUSCI_A2 eUSCI_A3 (UART, IrDA, SPI)
eUSCI_B0 eUSCI_B1 eUSCI_B2 eUSCI_B3 (I²C, SPI)

Figure 3.4: Block diagram of MSP432P401R launchpad

3.5 Bump Switches

Switches that respond to physical pressure or a "bump" are referred to as bump switches, tactile switches, or momentary switches. A brief electrical connection is made when the switch is depressed, which can be utilized to start a circuit or operate a device. Electronic gadgets frequently have bump switches as the user interface; by pressing the switch, users can operate the device. A bump switch can be used, for instance, to turn a device on or off, to switch between various modes or settings, or to initiate a particular activity.

Bump switches can be made for different levels of pressure sensitivity and come in a variety of sizes and shapes. When a bump switch is pressed, certain models incorporate a tactile feedback mechanism that emits a physical "click" or "snap" to let the user know the switch has been triggered.

Remote controls, keyboards, game controllers, and other electronic devices frequently employ bump switches. In industrial and automotive applications, such as in machinery and vehicle control panels, they are also utilized. Bump switches, which are frequently employed in a wide range of applications, are a basic but efficient technique to offer user input and control in electronic equipment.

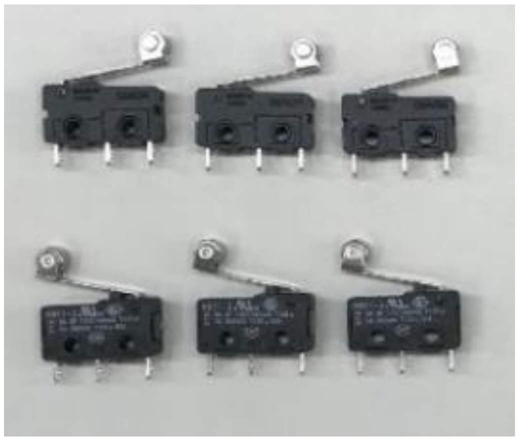


Figure 3.5: Six bump switches which are present on the board

The MSP432P401R development board has six bump switches. The MSP432P401R's GPIO pins can be linked to bump switches to offer user input and control for a variety of applications. The MSP432P401R has some built-in features that make interacting with bump switches simple, such as inbuilt pull-up and pull-down resistors that help maintain steady operation and minimize false triggers. The MSP432P401R also has a number of peripherals, like timers and interrupts, that can be utilized in conjunction with bump switches. For instance, a timer may be used to prevent the switch from being pressed repeatedly quickly, while an interrupt could be used to react to the switch push and initiate a certain action.

The MSP432P401R may be easily programmed to work with bump switches using a variety of programming languages and development environments. Code Composer Studio and Energia are just two of Texas Instruments' software development kits (SDKs) and integrated development environments (IDEs) that can be used to program the MSP432P401R.

3.6 Motors

An electromechanical tool called a servo motor is used to regulate the position, speed, and acceleration of a spinning shaft. Applications that demand precise motion control, such robotics, industrial automation, and aerospace, frequently use servo motors. A DC motor, a control circuit, and a feedback mechanism make up the fundamental parts of a servo motor. The servo motor is instructed where to go and how fast to move by the control circuit, which receives input signals from an external controller. A potentiometer or an optical encoder are frequently used in the feedback mechanism to track the position of the motor shaft and feed that information back to the control circuit.

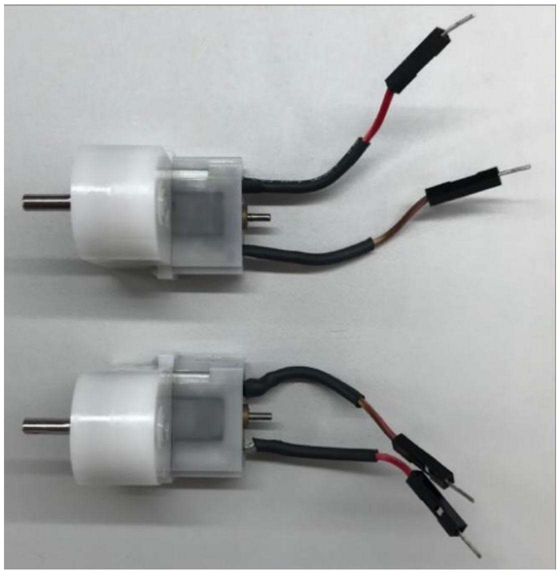


Figure 3.6: Motors used on the board

The ability of servo motors to hold an exact position even under shifting load conditions is one of their fundamental characteristics. Closed-loop feedback control is used to do this, where the control circuit continuously assesses the position of the motor shaft and modifies the speed and torque of the motor as necessary to maintain the desired position.

CHAPTER 4 : CODE COMPOSER STUDIO

4.1 Introduction

Code Composer Studio (CCS) is an integrated development environment (IDE) for software development with microcontrollers and embedded processors from Texas Instruments, such as the MSP430, C2000, Tiva, and Hercules families. A collection of software tools for creating, testing, and debugging embedded software are part of CCS, which is built on the Eclipse open-source platform. [3]

For the development of embedded software, CCS offers a complete set of tools, including an editor, a project management system, build tools, and a debugger. The editor supports a large number of programming languages, including C, C++, and assembly language, and offers features like syntax highlighting, code folding, and auto-completion.

Figure 4.1: Screenshot of Code Composer Studio opening window

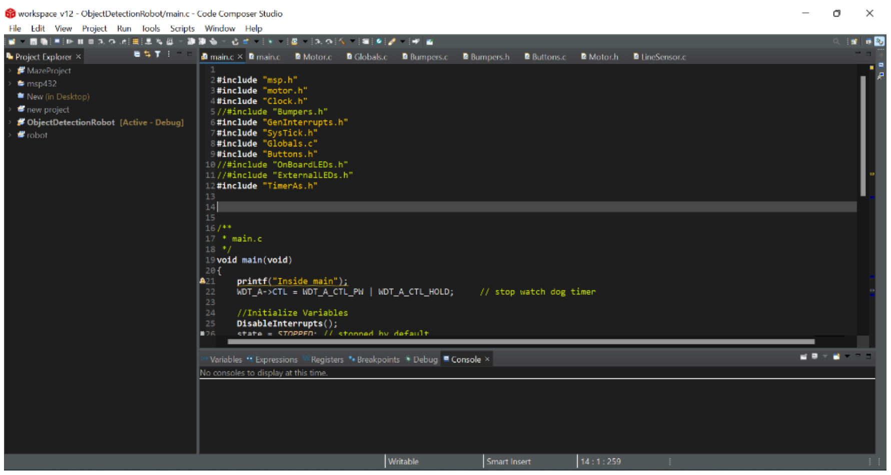
Developers can step through code, create breakpoints, and inspect variables and memory contents using CCS's robust debugging environment. Additionally, it includes real-time debugging features that let you inspect and alter the contents of registers and memories even as your program is still executing.

Software development kits (SDKs) and libraries for working with microcontrollers and embedded processors from Texas Instruments are included in CCS. They consist of sample code, drivers, and APIs that can be used as a jumping-off point for creating embedded software. [13]

In addition, CCS offers a range of extensions and plug-ins from outside developers that can be used to enhance the IDE's capability. For instance, CCS offers plug-ins for test automation, static code analysis, and version control systems. In general, CCS is a strong and flexible development environment for creating embedded software for microcontrollers and embedded processors from Texas Instruments. It supports a vast array of programming languages and hardware platforms and offers a complete set of tools for creating, testing, and debugging applications.

4.2 History of Code Composer Studio

Code Composer was originally a product made by GO DSP, a Toronto, Canada-based business that was eventually purchased by TI in 1997. Following the acquisition, Code Composer was given a real-time kernel called DSP/BIOS and had the term Studio attached to its name. [3]



```

#include "msp.h"
#include "motor.h"
#include "Clock.h"
//#include "Bumpers.h"
#include "GenInterrupts.h"
#include "SysTick.h"
#include "Globals.c"
#include "Buttons.h"
//#include "OnBoardLEDs.h"
//#include "ExternalLEDs.h"
#include "TimerAs.h"



/**
 * main.c
 */
void main(void)
{
    printf("Inside main");
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // stop watch dog timer

    //Initialize Variables
    DisableInterrupts();
    state = STOPPED; // stopped by default
```

No consoles to display at this time.

Figure 4.2: Screenshot of Code Composer Studio workspace

Up to version 3.3, CCS releases relied on a proprietary interface, but TI was already developing an IDE based on the open-source Eclipse at the same time. The MSP430 range of microcontrollers was the target market for this IDE, which went by the moniker Code Composer Essentials (CCE). Since the release of version 4.0, all versions of the previous CCS are also built on Eclipse thanks to the utilization of this expertise.

The availability of graphical visualization tools (XY graphs, FFT magnitude and phase, constellation, raw image visualization) and support for visualizing memory in a variety of numeric formats were two of Code Composer's key differentiators at the time because it was initially created for DSP development (decimal, floating-point). From 2015, a cloud-based version of CCS was released as part of the TI Cloud Tools package, which also includes Pinmux and Resource Explorer.

4.3 Features

Compiler - The C/C++ compilers in Code Composer Studio are designed to achieve the best performance and smallest possible code size for TI devices. Together with a compiler for microcontrollers based on the Arm® architecture, compilers for proprietary architectures like the MSP430TM, C2000TM, and DSPs are also offered. The LLVM and Clang compilers are used with additional TI technologies, like link time optimization, to produce code that is exceptionally small for TI Arm-based microcontrollers [11].

- Resource Explorer – It offers accessibility to the tools required for embedded development. Get examples, instructions, software development kits, and documentation quickly and specifically for the current platform.
- SysConfig - Pins, peripherals, drivers, radios, and other components can all be configured using the user-friendly and comprehensive utility known as SysConfig. SysConfig streamlines configuration issues and expedites program creation.
- EnergyTrace - An analysis tool called EnergyTrace allows an application be optimized for ultra-low power usage by measuring and visualizing its energy profile.
- Scripting and automation - A complete scripting environment provided by Code Composer Studio enables the automation of processes like performance benchmarking and testing.

4.4 System requirements

To install Code Composer Studio (CCS) a minimum of 4GB memory space and 2.5GB disc space is required along with a 2.0GHz single core processor. Although, for better performance it is recommended to have more than 8GB memory and 5GB disc space along with multi core processor. Code Composer Studio is a 64-bit application so will not run on operating systems with 32-bits.

4.4.1 Operating system

- Windows
  - Window 11
  - Windows 10 64-bit
  - Windows 7 64-bit
- Linux
  - Ubuntu 22.04 64-bit
  - Ubuntu 20.04 64-bit
  - Ubuntu 18.04 64-bit

4.5 Software Development Environment

The Eclipse open-source software platform serves as the foundation for Code Composer Studio. So, having a fundamental understanding of Eclipse will help you comprehend Code Composer Studio better. This is a description of some of the terms that are mentioned most frequently.

4.5.1 Workbench

The primary user interface is referred to as the Workbench. All the various views and resources used for development are available on the Workbench. The first dialog that appears when the Workbench is opened asks where the workspace will be. A single Workbench window appears after selecting the workspace location. A Workbench window provides a single or several viewpoints. Using the Window New Window menu, several Workbench windows can be opened. All Workbench windows refer to the same workspace and the same Code Composer Studio instance even though their visual layout (views, toolbars, etc.) may vary. The same project will be visible in all Workbench windows if it is opened from one Workbench.

4.5.2 Workspace

The workspace serves as Code Composer Studio's primary working location. Even if the actual projects don't exist inside the workspace folder, the workspace keeps references to all of the projects. New projects will be stored by default in the workspace folder. The Project Explorer view will display a project once it has been added to the workspace. When you run Code Composer Studio, a window asking for the location of the workspace folder will appear. To avoid being questioned again in the future, it is possible to request that the chosen folder be used as the default folder. Users' preferences and settings for the user interface are likewise kept in the workspace folder.

Workspaces are normally not shared by users and are user specific. To share your workspace with other team members, you wouldn't check it into source control. Your projects would be checked into source control, and each user's workspace would include references to the projects. In Code Composer Studio, only one workspace is open at once, but you can switch between them using the File --> Switch Workspace option.

4.5.3 Perspective

The arrangement of views, menus, and toolbars in the Workbench window is determined by a viewpoint. Each perspective offers a specialized set of tools for carrying out a particular kind of work. For instance, the CCS Edit perspective includes views like the Project Explorer, Editor, and Issues view that are frequently used during code development. Code Composer Studio will immediately convert to the CCS Debug perspective when a debug session is launched. By default, this perspective includes views related to debugging.

Using the viewpoint buttons at the upper right of the Workbench or the Window Perspective menu, one can manually change between perspectives. The perspective will retain any modifications made the next time it is opened. With the Window --> Perspective Reset Perspective menu, a viewpoint can be returned to its default configuration. Just storing the existing perspective under a different name will result in the creation of new perspectives. Save Perspective As... option under Window --> Perspective.

It is possible to access the CCS Basic viewpoint via the Getting Started view. The purpose of this perspective, which is used for both editing and debugging, is to make life simpler for users who are accustomed to simpler settings by only exposing the bare minimum of capability.

4.5.4 View

Inside the main Workbench window are windows called views that show information or data visually. The editor and a number of views are the major components of the Workbench window. The views Debug, Issues, Memory Browser, and Disassembly are a few examples.

4.5.5 Resources

Projects, folders, and files that are present in the Workspace are collectively referred to as Resources.

4.5.6 Project

Normally, projects include folders and files. A project corresponds to a real folder in the file system, similar to the workspace. On starting a new project, the default location is in the workspace folder, in a subdirectory with the project name. But you can also select a folder outside the workspace. The workspace will make a reference to the newly created project, which is then usable in the Workbench and accessible from the Project Explorer.

Projects can either be active or inactive. A project that has been closed remains in the workspace but cannot be changed by the Workbench. A closed project's resources still exist on the local file system even though they are not visible in the Workbench. Closed projects use less memory and aren't scanned when doing routine tasks. Hence, terminating pointless projects can enhance Code Composer Studio's performance. The Project Explorer will still display closed projects so that they can be quickly opened if necessary.

A project joins a workspace when it is either imported into the workspace or is created there.

All active workspace projects are displayed in Project Explorer. Keep in mind that the view mostly depicts the filesystem of the project folder. So, altering the actual file system occurs when a subfolder is created, and files are moved there from within the Project Explorer. The Project Explorer will display modifications made to the file system in a similar manner. It should be noted that not all files that appear in the view will also exist in the file system. Although linked files are references and not actual copies, they will show up in the view but not in the actual file system. It is not a physical folder, but the Included folder that shows in the Project Explorer displays all of the inclusion paths configured for the project.

4.5.7 File

A project can either have files added or linked to it. A file is copied to the project folder's root directory when it is added to a project. The option to "link" a file to a project is also available. Instead of copying the file into the project folder, this will just cause the project to generate a reference to the file.

4.6 Contents

A collection of development tools called Code Composer Studio includes an editor, a project management system, a compiler, a debugger, profiling tools, and visualization tools.

4.6.1 IDE

Integrated Development Environment is referred to as IDE. It alludes to a setting that assembles software development tools. Generally, an editor, build system, and debugger are involved. This eliminates the need to constantly switch between tools during software development.

A wide range of capabilities are available in the editor in Code Composer Studio to facilitate development. There are both more common features like local history and more uncommon ones like customized syntax highlighting and code completion. The local history records source code modifications and enables replacement of the current source with those from the history.

Building projects using the TI compiler or GCC is possible thanks to the project management system. It also interfaces with widely used source control programs like Git. The integrated debugger, which can be used to debug programs running on TI embedded devices, is covered in greater detail in the debugging chapter.

4.6.2 Compiler

There are C/C++ compilers available for every instruction set. This is typically a TI exclusive compiler. Although GCC is offered for Cortex A devices, it is generally advised to use the compiler that comes with the device's SDK. A TI proprietary compiler and GCC are offered for MSP430 and Cortex M based MCUs. The free-to-use GNU compiler is called GCC. The App Center offers a variety of compilers.

4.6.3 Resource Explorer

Resource Explorer makes it easy to locate the most recent datasheets, sample apps, libraries, examples, and more for the platform of your choice.

You can use the interface to filter the content to only what is pertinent to your chosen platform by selecting a device or TI LaunchPadTM Kit. Resource Explorer will display resources that are both installed locally on your computer and those that are accessible online and can be downloaded. Resource Explorer enables you to install the selected software package and any required dependencies before allowing you to import the selected example into your workspace when you choose an example from a software package that you have not yet installed.

4.6.4 App Center

App Center is used to acquire extra components needed for development and has a similar idea to Resource Explorer. App Center is generally used to obtain extensions or add-ons to the Code Composer Studio environment, such as compilers, as opposed to Resource Explorer, which is used to obtain software and documentation.

CHAPTER 5 : A* ALGORITHM

5.1 Djikstra Algorithm

Edsger W. Dijkstra, a Dutch computer scientist, created the Djikstra algorithm in 1956. The shortest path in a weighted graph can be found using Dijkstra's method, a graph search algorithm. The algorithm keeps track of both a set of visited and unvisited nodes. Apart for the starting node, which is added to the visited set, all nodes are initially unvisited. The method then continually chooses and includes in the visited set the node that is furthest from the starting node in the unvisited set [18].

Based on the edge weights and the most recent distance to the visited node, the method modifies the distance to every newly visited node's neighbors. The new distance is updated if it is less than the previous distance. When the destination node is added to the visited set or when all nodes have been visited, the process comes to an end. The path that the algorithm has constructed is the one that, at the conclusion of the procedure, leads from the starting node to the destination node in the smallest amount of time.

5.2 A* Algorithm

A* (pronounced "A-star") algorithm is a heuristic search algorithm used to determine the shortest path between any two nodes in a graph. Peter Hart, Nils Nilsson, and Bertram Raphael created it in 1968. By incorporating an additional heuristic function that calculates the separation between a node and a destination, the A* method improves upon Dijkstra's algorithm. The approach is more effective than Dijkstra's algorithm for big graphs because this heuristic is utilized to direct the search in the direction of the target node [19].

Two different sets - one of open nodes and one of closed nodes - are kept for the algorithm to function. The initial node is the lone element in the open set at first. The algorithm chooses the open set node with the lowest f-score at each iteration. The f-score is calculated by adding the node's g-score (which measures the actual cost from the starting node to the current node) and estimated h-score (the heuristic estimate of the cost from the current node to the destination) [20].

The algorithm assesses each node's non-closed set neighbors for each chosen node. The system calculates a preliminary g-score for each neighbor and updates its parent if the new g-score is lower than the old one. The neighbor is included in the open set if it is not already there. When the destination node is added to the closed set or when the open set is empty, the algorithm ends (since there is no way to go from the starting node to the destination). The path that the algorithm has constructed is the one that, at the conclusion of the procedure, leads from the starting node to the destination node in the smallest amount of time.

5.2.1 A* Algorithm in Robotics

The A* algorithm is frequently employed in robots for path planning and obstacle avoidance. Obstacle avoidance is the act of avoiding impediments in the environment while traveling along the intended path, whereas path planning is the process of determining a safe and ideal path from a starting point to a goal point. Robotics can benefit greatly from the A* algorithm's ability to handle complicated settings with obstacles and various paths while rapidly and effectively determining the best course of action.

The A* algorithm can be used in robotics to analyze a 2D or 3D grid map, where each cell represents a specific location in the surrounding environment. The algorithm works by representing the map as a graph in which each cell serves as a node and the edges between nodes are weighted according to the separation between nearby cells. A cost function that considers both the distance to the goal and the barriers in the environment must be defined in order to use the A* algorithm in robotics. The length of the path, the amount of time needed to travel it, or the amount of energy needed to move along it can all be used as the basis for the cost function.

Once the cost function has been established, the A* algorithm can be used to determine the best route between the starting and ending points, taking into account both the cost function and any environmental restrictions or impediments. The robot can then use sensors and other technologies to follow the intended course while dodging obstacles. Due to this, robotics applications like autonomous vehicles, drones, and industrial robots employ the A* algorithm extensively as a sophisticated path planning and obstacle avoidance tool.

5.3 Flow of A* Algorithm

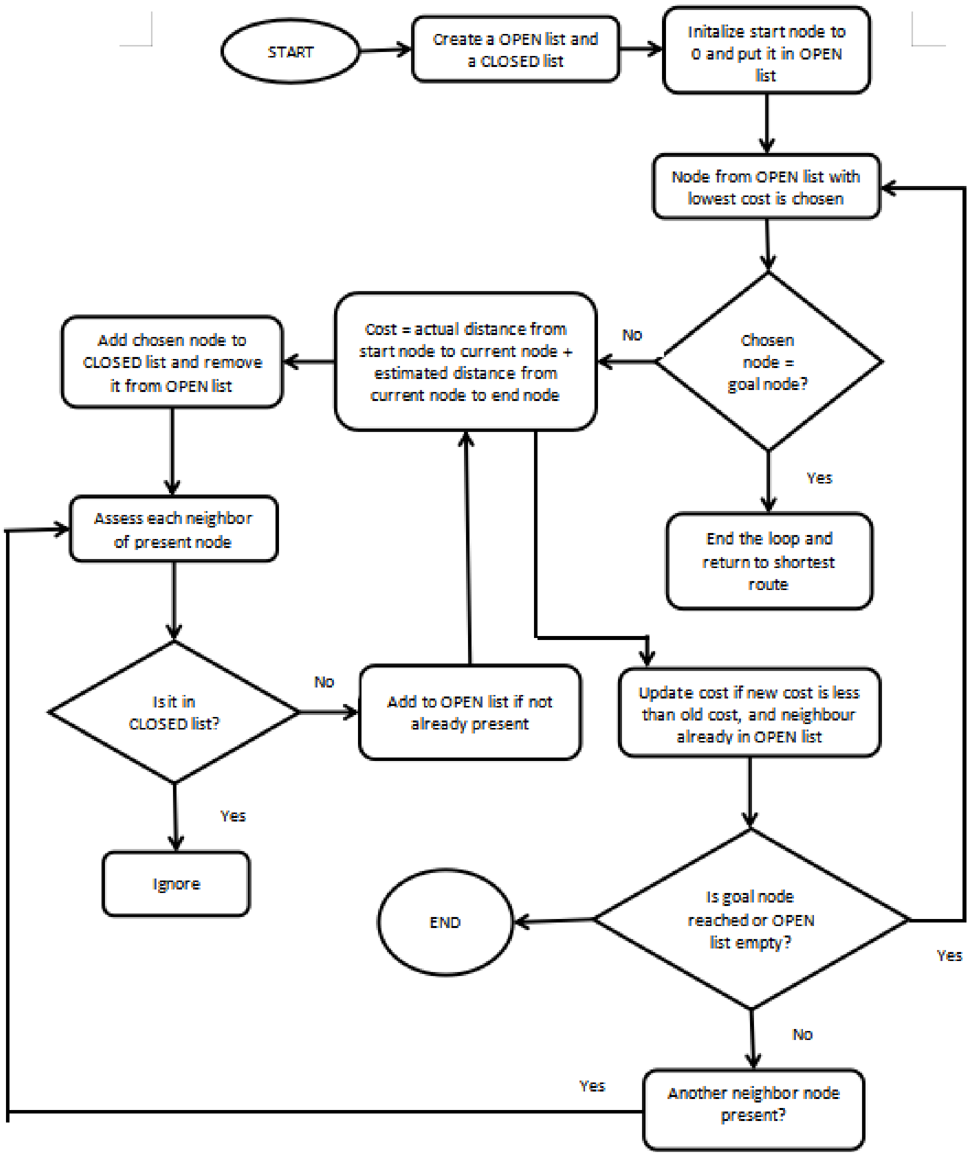
The goal node must be located, or the open set must be empty for the A* method to function. The algorithm's fundamental procedure can be summed up as follows:

1. To store the nodes that require evaluation, make an open set and a closed set.
2. Set the start node's cost to 0 and include it in the open set.
3. Continue until the desired node is located or the open set is empty:
   a. The node in the open set with the lowest combined cost should be chosen. This cost is calculated as the sum of the actual distance from the start node to the current node and the estimated distance from the current node to the goal node.
   b. End the loop and return the shortest route from the start node to the goal node if the chosen node is the goal node.
   c. Add the chosen node to the closed set and take it out of the open set.
   d. Assess each neighbor of the present node:
      i. Ignore a neighbor if it is a member of the closed set.
      ii. If a neighbor is not already in the open set, include it in the open set and set its cost to the total of the actual and estimated costs from the start node to the neighbor.
      iii. Update a neighbor's cost if the new cost is less than the old cost and the neighbor is already in the open set.
4. There is no path from the start node to the target node if the goal node cannot be located and the open set is empty.

The A* algorithm ultimately chooses the node with the lowest total cost, assesses its neighbors, and updates the open set with the cost of the best path thus far. Once the goal node is reached or the open set is empty, indicating that there is no path to the goal node, the process is repeated.

5.4 Flowchart of the principle of how A* Algorithm works

Given below is the flowchart explaining how the A* algorithm functions at every step and what action it takes when certain conditions occur.



START
Create a OPEN list and a CLOSED list
Initalize start node to 0 and put it in OPEN list
Node from OPEN list with lowest cost is chosen
Chosen node = goal node?
No
Yes
End the loop and return to shortest route
Cost = actual distance from start node to current node + estimated distance from current node to end node
Add chosen node to CLOSED list and remove it from OPEN list
Assess each neighbor of present node
Is it in CLOSED list?
No
Yes
Ignore
Add to OPEN list if not already present
Update cost if new cost is less than old cost, and neighbour already in OPEN list
Is goal node reached or OPEN list empty?
Yes
No
END
Another neighbor node present?
Yes

Figure 5.1: Flowchart showing how A* algorithm works

As seen in the flowchart above, the A* algorithm will search for open nodes among its neighboring nodes and find the node with shortest distance to the goal node by calculating the distance of all nodes on the path.

CHAPTER 6 : IMPLEMENTATION

6.1 Making the maze

To check if the project is successfully working, a maze needs to be designed. This is done with the help of wooden planks. Designing the correct type of maze is important for the project to be implemented successfully. The first thing that needs to be decided is the shape and size of the maze. Maze can be rectangular, pentagonal or even a circle. In this case, a rectangular maze was made. The size of the maze was 6 inches * 6 inches.

After determining the size and shape, a prototype of the maze needs to be drawn in order to make it easy to construct the maze. The first thing that needs to be determined is the starting point and ending point which can be at any location in the maze. Once they are determined, the rest of the maze can be drawn starting with the wall on the outside of the maze. After the outer layout is ready it becomes easy to draw the inner walls.

Once all the walls are drawn, barriers need to be added. This can include creating dead ends or placing an object in between to block the robot's path. But this should be done carefully as adding too many obstacles in the same part of the maze may make it very difficult for the robot which may lead to it taking a very long time to reach the end point. Thus, the barriers need to be created strategically by spreading them uniformly throughout the maze.

After the maze is drawn, it should be checked whether it is solvable and should have at least one path from start point to end point without any obstacle. The difficulty needs to be determined and accordingly changes need to be made if required. The important thing here is that the difficulty should neither be too easy for the robot to solve it very quickly, nor be too difficult that the robot would fail to solve the maze. There must be a fine balance between the two to get an optimal result. Once the layout is made successfully, it can be constructed using wooden planks. The path in the maze should be wide enough for the robot to pass comfortably without getting cramped or hitting the wall on the sides.

6.2 Developing the code

As explained from the previous parts, the goal of the algorithm is to find the optimal solution to the maze. The robot was trained to travel freely into the maze to get to know the layout and form a virtual map into its system. It will try to take into account information regarding the turns it comes up against, and this knowledge will help it to remember the next time it comes at the same spot or at a similar looking turn. Cost of every node is calculated until there is no other way in the node or if it reaches the goal. At every turn, the costs are compared and the lower one is determined.

In this step, I faced some issues. The robot would drift a lot and wouldn't go in a straight line, it would bump on the side walls. I am assuming the cause is the failure of the wheel to adapt to different surfaces as the wheels wouldn't hold its shape on hitting against the wall. This resulted in the robot losing its momentum and due to it, when it would hit the wall, it won't take a prefect 90 degree turn; it would be little bit less than that (around 70-75 degrees). As a result, on occasions it would keep moving in the same space due to the fact that it wasn't moving in a straight line. As a result, on some occasions, the costs wouldn't be updated when in theory, they should have. Therefore, even for a very small maze, the robot would take a very long time to find its way out.

To solve this issue the robot was first trained to make its way through the maze without taking into consideration the costs. So it would just travel freely and at every wall keep on taking turns. This helped the robot to make better turns as it would just have to keep moving without having to compare costs and decide on its next move. This resulted in good improvement in terms of the movement of the robot as it reduced the times when it would hit the side wall.

The robot is allowed to enter the maze after pressing the wait button, it will start moving forward until it comes across an obstacle, in this case a wall. While coming up against a wall, it will have two options – to turn left or to turn right. In such a scenario, the robot will stop and move a little bit behind and then take the appropriate turn. Here, the robot is coded such that priority is given to turning left. So, whenever a situation comes when both turns are present, the robot will take a left turn.

Three separate functions are created – forward, backward and left function. These functions are primarily designed to control the two motors in this project. Each function has two important

variables – one for setting its direction and the other for setting its speed. Motor being set in whichever direction will determine which way it will move. In the forward function, both the motors are set to move in the forward direction, and both are set to have the same speed. While in the backward function, both the motors are set to move in the backward direction. In the left function, the objective is to turn left and to achieve this one motor should be moving forward and the other should be moving backward direction. So, to move left, the left motor is set to move backward, and the right motor is set to move forward. If the motor must take a right turn, then the opposite should be done. Also, to keep the motor moving in a straight line, it is imperative that both the motors are set at the same speed. The forward function has a speed set to 15% while backward function has a speed set to 14%. The backward function has slightly less speed because we only have to move the motor a little bit backward while taking a turn. A small delay is added in the backward and left function to allow a small window for the robot to process the next step as well as to make it easy for the viewer to understand what is happening. The speed range of the robot is set as 0-100.
In such a way, the robot would be able to navigate its way through the maze. This can be thought of as a trial-and-error way to solve the maze. Due to the issue faced I decided to create a small maze where the turns are at a short distance to offset the chance of bumping on the side wall. But still at times when the robot would hit a dead end it would fail to make its way out as it would be surrounded by wall on three sides so if it wouldn’t be able to take an exact 90 degrees turn it wouldn’t be able to make its way back. So I decided to make the demo video with a straightforward maze which just shows the robot making its way out of the maze without any dead ends.

The board is coded in such a way that at the first obstacle the robot will take a 90 degree turn to the left. But in the scenario that after taking a left it immediately hits a wall, then if it again takes a 90 degree left turn, it would mean that the robot took a total 180 degree turn, meaning it will go back to the start point. To solve this issue, the robot is coded such that in such a scenario, it will take a 180 degree turn. This would mean that the robot is taking a right turn from the original point when it comes up against a wall in the first place.

Due to wheel resistance after getting in contact with the surface, at times the robot would start drifting little bit towards left or right. To overcome this, the two left most sensors were coded to take a right turn of 20 degree when being pressed. Also, the two right most sensors were coded to take a left turn of 20 degrees after coming in contact with the wall.

After developing the source code for the project, it needs to be stored on the MSP432P401R development board. The board has a USB port to carry this out. The code first needs to be checked for any errors which will be shown in the console. These errors need to be resolved before the code is built. After the source code is ready and is finished building, it needs to be debugged. This is done by connecting the board to the Code Composer Studio on the computer with the help of cable provided in the kit.

To turn on the motors on the board, six AA batteries are needed. After turning on the board, the green LED will be turned on which will indicate that the board is on. While the debugging process is carried out, the red LED will keep on blinking, indicating that the code is being fed on to the microcontroller. After that is done, the LED will stop blinking and will give a solid red output. This means that the code has been successfully debugged onto the development board. There is another switch near the main switch which needs to be pressed. This will turn on the motors and when placed on a flat smooth surface, will enable the robot to move.

CHAPTER 7 : RESULTS AND CONCLUSION

7.1 Result

After the robot is kept at the starting point and is turned on, it moves forward until it comes across a wall. Due to some issue of wheel resistance, the robot does not go in a perfect straight line. I tried many ways to fix this but was not able to completely fix this issue. To make up for this, some changes had to be made which slightly minimized the use of A* algorithm but it was necessary to get the result.

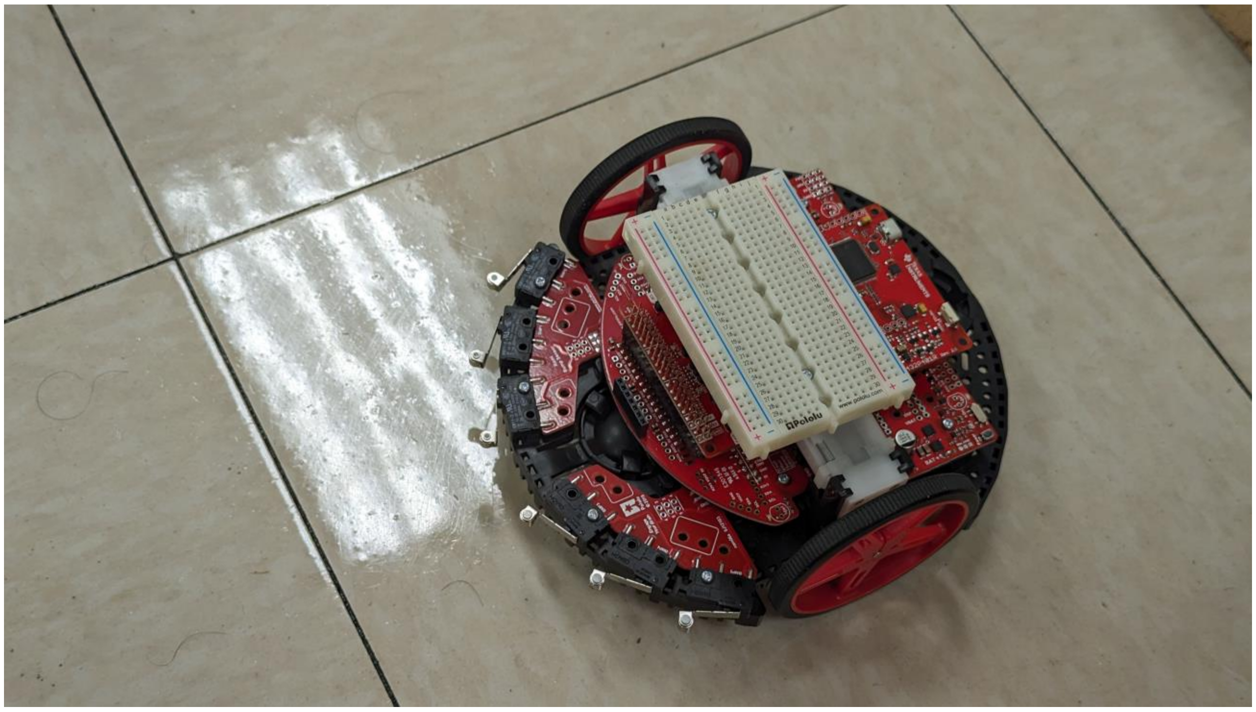


Figure 7.1: The maze solving robot

The bump sensors attached to the front of the robot were successful in detecting walls and obstacles which came across it and as the result played a crucial part in solving the maze.

7.2 Conclusion

This project shows that artificial intelligence can be successfully used for maze solving which can turn out to be an important step on the path of artificial intelligence taking over the world in the next few years. The learning kit was new to me, so I needed to change a few things from the original plan to execute this project successfully. There is still a lot of scope for improvement in this area, especially in regard to enhancing the navigation and obstacle avoidance accuracy and effectiveness of the robot. Overall, maze-solving robots have a bright future, and we may anticipate additional developments in this field in the years to come.

7.3 Future Scope

Robots that can navigate mazes have a huge range of possible uses in a variety of industries. There are several potential applications for these robots. Maze-solving robots can be employed in search and rescue operations to find and save individuals in need by navigating through challenging terrain like buildings or rubble. This can help save a lot of time and potentially save lives. Robots that can solve mazes can be utilized in manufacturing and logistics to efficiently transfer products and supplies through factories or warehouses.

We may anticipate seeing more effective and intelligent maze-solving robots in the future that can carry out more difficult jobs with increased accuracy and precision thanks to the growing advances in robotics technology. Artificial intelligence and machine learning can also be used to improve the capabilities of these robots and increase their adaptability to various settings and tasks. Overall, maze-solving robots have a bright future, and in the years to come, we may anticipate more fascinating advancements in this area.

In my particular project, if the issue of wheel resistance needs to be worked on which would make it successful in solving bigger mazes without any difficulties. This can be helped if one develops motors which are adaptable to the surface such that when the surface is very smooth, it would still help the wheels to hold their shape and not lose its momentum.

REFERENCES

[1] *MSP-EXP432P401R: Out of Box* MSP-EXP432P401R | Out of Box - MSP-EXP432P401R - Welcome 1.20.00.45 documentation. (n.d.). Retrieved April 7, 2023, from https://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/SIMPLELINK_MSP432_SDK/1.20.00.45/exports/docs/simplelink_mcu_sdk/project0/project0/docs/MSP-EXP432P401R.html

[2] 2.1. System Requirements - Code Composer Studio 12.2.0 Documentation. (n.d.). Retrieved April 7, 2023, from https://software-dl.ti.com/ccs/esd/documents/users_guide/ccs_overview.html

[3] Wikimedia Foundation. (2023, June 13). *Code composer studio*. Wikipedia. https://en.wikipedia.org/wiki/Code_Composer_Studio

[4] Maze Solving Robot with Automated Obstacle Avoidance Redirecting from https://doi.org/10.1016/j.procs.2017.01.192

[5] Survey on techniques used in Autonomous Maze Solving Robot *IEEE Xplore*. from https://ieeexplore.ieee.org/Xplore/guesthome.jsp

[6] Autonomous Maze Solving Robot *IEEE Xplore:* Advanced Search from https://ieeexplore.ieee.org/search/advanced

[7] Alamri, S., Alamri, H., Alshehri, W., Alshehri, S., Alaklabi, A., & Alhmiedat, T. (2023, February 8). An Autonomous Maze-Solving Robotic System Based on an Enhanced Wall-Follower Approach. Machines; MDPI. https://doi.org/10.3390/machines11020249

[8] An Efficient Algorithm for Robot Maze-Solving. (2010, August 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/document/5591159

[9] ArcBotics - Maze Solving. (n.d.). http://arcbotics.com/lessons/maze-solving-home-lessons/

[10] Learn How Artificial Intelligence (AI) Is Changing Robotics. (n.d.). Intel. https://www.intel.com/content/www/us/en/robotics/artificial-intelligence-robotics.html

[11] CCSTUDIO IDE, configuration, compiler or debugger | TI.com. (n.d.). https://www.ti.com/tool/CCSTUDIO

[12] Sudhakara, P., & Ganapathy, V. (2016). Trajectory Planning of a Mobile Robot using Enhanced A-Star Algorithm. Indian Journal of Science and Technology, 9(41). https://doi.org/10.17485/ijst/2016/v9i41/93816

[13] Zidane, I. M., & Ibrahim, K. (2017). Wavefront and A-Star Algorithms for Mobile Robot Path Planning. Advances in Intelligent Systems and Computing. https://doi.org/10.1007/978-3-319-64861-3_7

[14] Hackster.io. (2019, May 13). AI Maze-Solving Robot. https://www.hackster.io/crescenters/ai-maze-solving-robot-ac21f8

[15] Wikipedia contributors. (2023, March 28). Microcontroller. Wikipedia. https://en.wikipedia.org/wiki/Microcontroller

[16] Wikipedia contributors. (2023a, March 10). Maze-solving algorithm. Wikipedia. https://en.wikipedia.org/wiki/Maze-solving_algorithm

[17] Wikipedia contributors. (2023b, March 27). Integrated circuit. Wikipedia. https://en.wikipedia.org/wiki/Integrated_circuit

[18] Wikimedia Foundation. (2023a, May 28). *Dijkstra's algorithm*. Wikipedia. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[19] Wikimedia Foundation. (2023c, July 9). *A* search algorithm*. Wikipedia. https://en.wikipedia.org/wiki/A*_search_algorithm

[20] GeeksforGeeks. (2023, March 8). *A* search algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/a-search-algorithm