



Authors:

Albert Lougedo
*Electrical
Engineering*

Matthew Carvajal
*Computer
Engineering*

Mahyar Mahtabfar
*Electrical
Engineering*

Reviewer Committee:

Dr. Reza Adbolavand
*Professor
Electrical
Engineering*

Dr. Sonali Das
*Professor
Electrical
Engineering*

Dr. Mikhael B. Wasfy
*Professor
Electrical
Engineering*

Advisor:

Dr. Wei Lei

January 24th, 2025



1. Introduction / Narrative

1.1 Project Background

Humans are unquestionably the smartest beings on the planet. From the dawn of time there have been nonstop innovations stemming from fire to the world around us today. However, with such innovation stem new challenges to overcome. Technological advancements have come to aid humanity to live a more efficient life and spark innovation. In recent decades, one particular invention came to fruition, the SmartPhone. The SmartPhone has made communication via the internet possible with the combination of the cell phone which came decades prior. Being such a monumental piece of technology, we as a species have become reliant on them, looking at notifications for communication or browsing the internet for necessary information all at our fingertips. With such amazing technology came one of our biggest flaws, distractions. As consumers of smartphones, we are all susceptible to having our attention diverted by applications designed for our amusement and to focus on areas of non-importance. Such distractions that stemmed from the smartphone are social media, messages and games. These distractions get in the way of everyday life as we know it, continuing to shorten the attention span of the average consumer. Whether it be in a classroom, at work, cooking dinner or even waiting at the doctor's office, the smartphone and its captivating applications have infiltrated our daily lives at a grand scale. One daily task that it has invaded is commuting.

One of the biggest hassles people face on a day to day basis, often even multiple times a day, is commuting in their vehicle. In this day and age a normal traffic stop at a red light makes the general population believe there is enough time to catch up with their friends and family on popular social media apps like Instagram and Facebook. As quickly as they may perceive their phone usage at the red light, drivers often feel in a state of comfort when driving, oftentimes not remembering specific events on the road and just recalling arriving at their destination. This state of comfort leads directly to a lack of importance perceived from the driver and then in turn the brain looks for its favorite source of dopamine: the smartphone and the endless applications it holds.

After spending some time in traffic and looking around at the other drivers at a stop, nearly all drivers are tinkering on their cell phone whether it's changing a song, sending a text or responding to an email. When the world's drivers are constantly distracted from the traffic around them, this directly leads to a higher cause of accidents between vehicles and a higher danger to pedestrians. The goal with this product is to reestablish attention and bring awareness back to safety on the roads. Alongside, big automobile manufacturers have made efforts to keep the driver engaged while on the road in latest models. Many of the automotive companies like Hyundai and Tesla for example are implementing new sensors in their cars to encourage drivers to drive as intended rather than cause unnecessary delays. Many of these sensors include forward collision warnings, lane departure warnings, lane centring assistance, and adaptive cruise control. Even though these technologies are made for aiding in the safety of the roads, they do not address the lack of attention that is growing in our society.

What if there were a compact device, no larger than a standard transponder, designed specifically to help drivers maintain focus on the road? Hence, came the idea of **SafeSight**, a device that is able to track traffic light changes, when the car in front of you

is leaving, and take photo documentation of a trip so that drivers can realize how distracted they can be when commuting.

1.2 Project Motivation / Current Commercial Technologies

The motivation of **SafeSight** is to develop a device that brings the attention of the driver back to the roads, aids lack of response times, and helps ensure safety on the roads for commuters. SafeSight is a tool that leverages the advancements of modern technology and combines the existing technology in a matter to bring the integrity and efficiency of the roads back to a higher level than that of the time prior to smartphones.

For SafeSight to truly live up to the ambition we have for it, the team has come together to develop it in a way that allows for ease of access and compatibility with modern technologies. For instance we have limited the size of this device to be the size of a transponder that many individuals already carry in their vehicle for toll purposes. The technology in the device is also compatible with applications to store data from a commute to a user's profile in order to encourage better driving practices whenever the user returns on the road. According to a recent study, 3,047 fatal crashes were caused by distractions. This number alone accounts for 8% of all fatal crashes in the United States. In addition to these fatalities, there was a total of 362,415 people injured from distracted drivers. Now, not all of these distractions are caused by smartphones, but rather a vast majority of them are. For this project to really be effective, the device would combat any distraction, alerting the driver and reducing the number of fatalities we see from such a preventable occurrence.

As we mentioned earlier, there are some technologies that do in fact help ensure safety but they do not attack the distractions. Moreover, these technologies are only installed into vehicles as of late, leaving nearly a century of automotive engineering without safety features. Our goal is for this device to be able to be implemented on every single vehicle, offering a much more intuitive and updated experience for every make and model on the road today. SafeSight will make the commuting process much more efficient and in turn reduce accidents, commute times, and unexpected obstacles from distracted drivers.

1.3 Project Function

SafeSight will utilize computer vision to classify and make connections between the driver and their attention towards the road. Cameras will continuously monitor the driver and outside traffic signals to then alarm the driver if they are not paying attention to the road. For example, a driver is caught at an intersection waiting for the traffic signal to switch to green. They suffer from a short attention span and resort to their mobile device to pass the duration of the red light. This distraction prevents them from properly reacting to the traffic signal switching to green, causing unnecessary delays and inefficient flow of traffic. These attention infractions made by the user of the vehicle will be acted upon by a sound alarm and recorded by the Safe Sight device. Safe Sight, in a manner, will make its user's driving more efficient by increasing response time, reducing error when driving, and maintaining alertness to surroundings. SafeSight will actively track the user's performance on the road by documenting how many times the user committed an attention infraction. The device will communicate with the user by projecting an audio signal outside of a speaker connecting to the PCB. This audio signal will serve as a sensory reminder to get off the phone or divert attention back to the road.

Additionally, the computer vision will be trained to classify if the driver in the vehicle is looking down while the vehicle is in motion, where the device can then register and act upon an attention infraction. Our team must design a system that is both lightweight and compact in order for the device to function as anticipated, but also there comes the hardware necessary to make these ambitions come true.

2. Project Objectives, Requirements, and Goals

2.1 Project Objectives

SafeSight will encompass a system that brings back driver integrity and safety to the traffic environment as a whole. With this comes an increase in efficiency from the traffic system as well, leading to reduced commute times with reductions in accidents and injuries caused on the roads. When a commute begins, the system should be started and acquire traffic data to give a numerical representation of a driver. From every traffic stop, lane merger, and cruising scenario, the SafeSight system will constantly provide feedback to keep the driver alert. This same system should also detect changes in a driver's performance, for example if a driver is appearing to swerve between lane lines, then the device will suggest taking a break due to restless sleeping habits. If the driver is taking longer than usual to respond to changes in a traffic light, the device will notify the driver to focus their attention back on the road. These behaviors can be recognized by the SafeSight system in order to properly assess the reason for the driver's lack of performance.

SafeSight should encompass a system that captures the current state of a driver and multiple sample points during a journey. This system will then process the world around the host vehicle and the data collected from the driver's performance to assess recommendations (if any) to the driver on how to be more safe and efficient on the roads. The system in itself should be able to detect pedestrians, change in traffic light patterns, stop signs, movement from the host vehicle, and distances from cars ahead with the help of computer vision algorithms. These algorithms will be tested through the hardware of the Raspberry Pi and the tests will be deemed successful if it efficiently identifies the change in traffic light colors where red = stop and green = go. The system should also correctly sound the auditory signal when the driver is idle for a longer than anticipated reaction time to a vehicle moving ahead of it. SafeSight should also be able to identify pedestrians crossing and offer a signal that the driver must come to a stop, this scenario is also expected at a stop sign.

SafeSight should encompass a system that knows when the driver should be ready to progress their journey or not. This will require for the system to be able to detect movement in the vehicle, notify the driver for longer than anticipated stop times.

All of the components of SafeSight should be modular, meaning that each component should have the ability to be individually tested prior to being integrated to the entire system. The components of SafeSight should be able to be assembled and should **not** be able to be disassembled quickly, as it needs to adhere to conditions of a moving vehicle and portable. The entire system should be able to fit on the windshield of a vehicle without obstructing the driver's view.

SafeSight is currently a self-funded project, and with that comes the ambition to make sure it is cost effective for someone to replicate themselves if another engineer found interest in doing so. For the lack of sponsorship, the team has an aim to make this device for under \$300, so that each member does not contribute more than \$100. In the scenario where significant changes are needed, the budget shall be adjusted accordingly.

2.2 Goals

❖ SafeSight Goals:

➤ Basic

- Maintain drivers attention to the road at all times while the vehicle is in motion by setting off an alarm if the driver deviates their attention from the road. CV will be trained to classify if a driver is looking at the road or looking down (at their phone).
- Analyze common traffic lights and alert the driver if they are not following them properly. Train CV to classify, depict the contours, and differentiate colors of common traffic lights.
- Establish connectivity with all components of the final product: PCB and cameras.
- Train CV and analyze/stream video at 15-30 frames per second
- Using an established UART connection with the raspberry pi. Receive inputs from the computer vision indicating attention infractions and send an output signal to a speaker connected to the PCB board to alarm the driver.
- Read register values from the inertial sensor to dictate whether the car is in motion or at a stop.

➤ Advanced

- Develop an application that will pair with the device to pull driver data that is collected along trips. For example, an application on a user's phone will keep a tally and download pictures of attention infractions that the user makes.
- Take pictures using the cameras and stream video to the application for it to be accessed by the user.
- Implement interrupt service routines to have the MCU operate in a low-power mode to conserve energy. MCU will be interrupted by an attention infraction and then alarm the driver of the vehicle by outputting an alarm signal to a speaker.

➤ Stretch

- Create a pre-collision alarm for the driver to notify them of imminent danger on the roadway. Utilize CV in a fast paced environment to depict the speed of incoming objects relative to the vehicle. Then, make a decision to alert the driver to react if they have not been responsive to their surroundings.

2.3 Requirement Specifications and Constraints

The previous sections in this report have described the ambitions and concepts encompassing SafeSight. In order to bring these ambitions into fruition, there are a certain amount of required specifications we must adhere to both hardware and software based. These requirements are what the SafeSight team believes are necessary to bring the project to its fullest potential. Let these requirements serve as a meeting ground for the members of this team and the advisors to all understand the goals and aspirations for what this project will be able to accomplish.

Initially, the first constraint that comes to mind with a compact mounted design is power. How will we power a raspberry pi computer and MCU executing real-time processing tasks including computer vision? Well, first we need to calculate roughly how much power in watts per hour will our execution of safesight require. Below are the average power requirements for the raspberry pi and ESP32 running at various demands.

ESP-32 MCU

Raspberry Pi 4 B

Pi State	Power Consumption
Idle	540 mA (2.7 W)
ab -n 100 -c 10 (uncached)	1010 mA (5.1 W)
400% CPU load (stress --cpu 4)	1280 mA (6.4 W)

Power mode	Description			Power Consumption
Active (RF working)	Wi-Fi Tx packet			Please refer to Table 5-4 for details.
	Wi-Fi/BT Tx packet			
	Wi-Fi/BT Rx and listening			
Modem-sleep	The CPU is powered up.	240 MHz *	Dual-core chip(s)	30 mA ~ 68 mA
			Single-core chip(s)	N/A
		160 MHz *	Dual-core chip(s)	27 mA ~ 44 mA
			Single-core chip(s)	27 mA ~ 34 mA
		Normal speed: 80 MHz	Dual-core chip(s)	20 mA ~ 31 mA
			Single-core chip(s)	20 mA ~ 25 mA
Light-sleep	-			0.8 mA
Deep-sleep	The ULP coprocessor is powered up.			150 μ A
	ULP sensor-monitored pattern			100 μ A @1% duty
	RTC timer + RTC memory			10 μ A
Hibernation	RTC timer only			5 μ A
Power off	CHIP_PU is set to low level, the chip is powered down.			1 μ A

Table 5-4. Current Consumption Depending on RF Modes

Work Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	—	240	—	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	—	190	—	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	—	180	—	mA
Receive 802.11b/g/n	—	95 ~ 100	—	mA
Transmit BT/BLE, POUT = 0 dBm	—	130	—	mA
Receive BT/BLE	—	95 ~ 100	—	mA

We need to choose the values that assume we are running every component in its most energy demanding mode so that when we choose our power source, all processes will be properly powered to assure performance of all components. Using the formulas below we can calculate our most demanding average power in watts per hour.

$$\text{Average Power Consumption (Watts Per Hour)} = V(\text{Voltage}) * I(\text{Current in mA})$$

$$\text{Total Power Supply (Watts)} / \text{Average Power Consumption (Watts per Hour)} = \text{Operational Hours}$$

The values chosen for our raspberry pi will be 5V and 1280mA, demanding 6.4 watts per hour. Additionally, our ESP-32 will demand 3.3V and 240mA which makes about 0.8 watts per hour. Finally, we can calculate our average power consumption and operational hours. If Battery powered, we will assume a 5V 10000mAh battery.

Total Power Supply = $5 \times 10 = 50$ Watts **Average Power Consumption** = 7.2 Watts/Hour

Operational Hours = $50 / 7.2 = 7$ Hours

Now that we have documentation on the requirements and performance of powering our product, we can conclude that a battery powered method of power delivery will not provide convenient usage of the product for the user due to frequent recharges needed from driving, roughly every 7 hours. However, our solution to this is to use the 12V charger port in the vehicle with a voltage converter circuit to properly power the device with no worry of any recharge. The wired design will drop our product in size and weight. Furthermore, Most car charger ports run 12V with a 10A fuse which provides 120 Watts of power! This is more than enough power that our device needs and will be the power source tapped into by our Safe Sight product.

2.4 Engineering Specifications

Specifications for each of the project's subsystems are listed below:

Model Specifications:

- The device cannot be bigger than 8.5 inches x 7 inches to abide by federal highway safety regulations
- Should weigh less than 900 grams

MCU Specifications:

- Should power on the rest of the system when the 12V source is connected to the device
- Amplifies the excitatory signal from the raspberry pi and inertial sensor to the pcb speaker's requirement of 50-100mV RMS.
- Properly transmits the excitatory signals from both the CV (raspberry pi) and the sensor around the rest of the system with the base UART rate of 5 Mbps.
- Maintains a connection to the user's device that is housing the software via bluetooth from 1.5KB/s to 10KB/s.

RaspberryPi Specifications:

- Properly utilizes the CV algorithms to identify traffic distances and changes at a processing power of 2.4GHz.
- Captures and stores the instances when a driver is distracted via Image Signal Processor and transmits the data to the mcu.
- Executes the two cameras on their respective CV algorithms with no computational overlap with dual 4k display output.

CV Algorithm:

- Identifies the change in traffic conditions such as a moving vehicle, distance, and light changes.
 - Color detection through color thresholding
 - Distance calculation from size change analysis
- Facial recognition to detect when the driver is not looking straight ahead at the road, OpenCV.

Inertial Sensor Specification:

- Detect when the host vehicle has reached a net velocity of 0.
- Detects when the host vehicle has begun movement, immediately signaling the end of an excitatory signal to the MCU when inertia >0.

Software Specifications:

- Should be able to power on immediately when plugged into the 12V source.

Table 2.4.1 Hardware Component Specifications

Component(s)	Parameter	Specification
Application	Distraction tracking / Live camera feed	Tracks number of distractions and shows live camera feeds
Cameras, raspberry Pi, sensors	Car, light and motion detection	CV algorithms for color thresholding, size change analysis, and face orientation recognition.
Mini adafruit speaker	Beep sound/Action Delay	102dB +/- 3dB , 600 Hz to 10kHz
Power	Cigarette lighter adapter	12V
Led	To show what light is being detected	1-2 Lumens
Microcontroller ESP 32	Performs calculations and communicates with Pi. Transmits data to the application host device.	Data transfer >= 5Mbps
IC (integrated circuit)	Non-inverting Operational Amplifier to perform message signal amplification.	Gain of range from 50-100mV

Table 2.4.2 Tentative Bill of Materials

Component	Quantity	Unit Cost	Total	
Adafruit BNO055 (Inertial Sensor)	1		\$35	
Raspberry Pi v5	1		\$90	
Raspberry Pi HQ Camera	1		\$50	
Micro SD	1		\$12.00	
Raspberry Pi camera	1		\$35	
LED	1		\$3	
Voltage Regulators	1		\$5	
ATMega 328P (MCU)	1		\$3	
PCB Board	1		\$30	
PCB Speaker	1		\$2	
Signal Amp I.C	1		\$4	\$269

2.5 System Diagram and Visualization

In an effort to bring our ideas to fruition we have drafted a visual representation of both the hardware and software of the system as a whole. These diagrams highlight not only the flow of our system but also references the engineers who are taking responsibility for overseeing the specific components in the system. All of the engineers who are working on Safesight will make equal contributions on all systems and their subsystems, however in an effort to diversify the workload and abide by the constraint of time, assignments to specific areas were made.

Table 2.5.1 Software Block Diagram

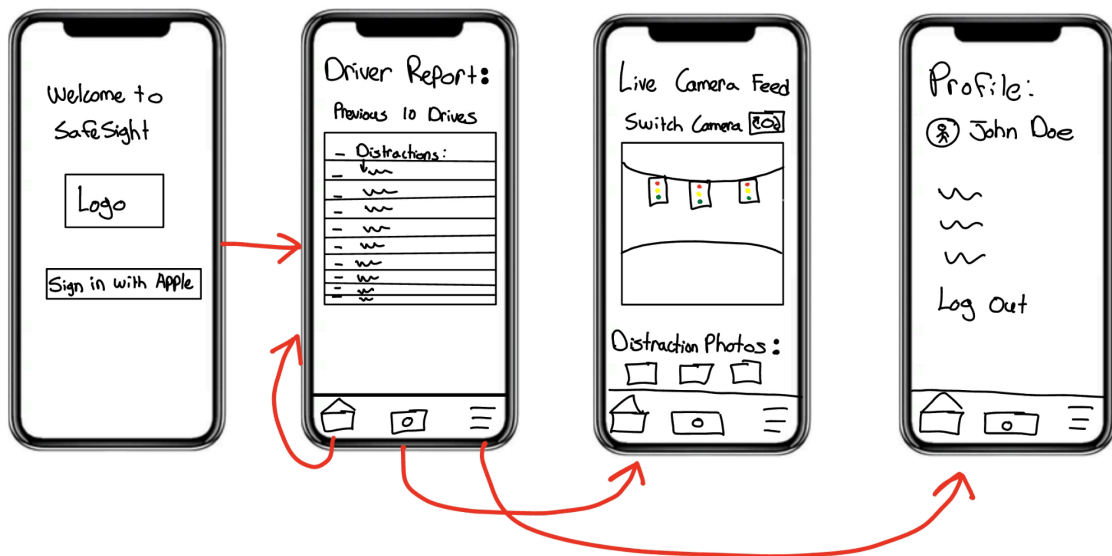


Table 2.5.2 Software Block Diagram(2)

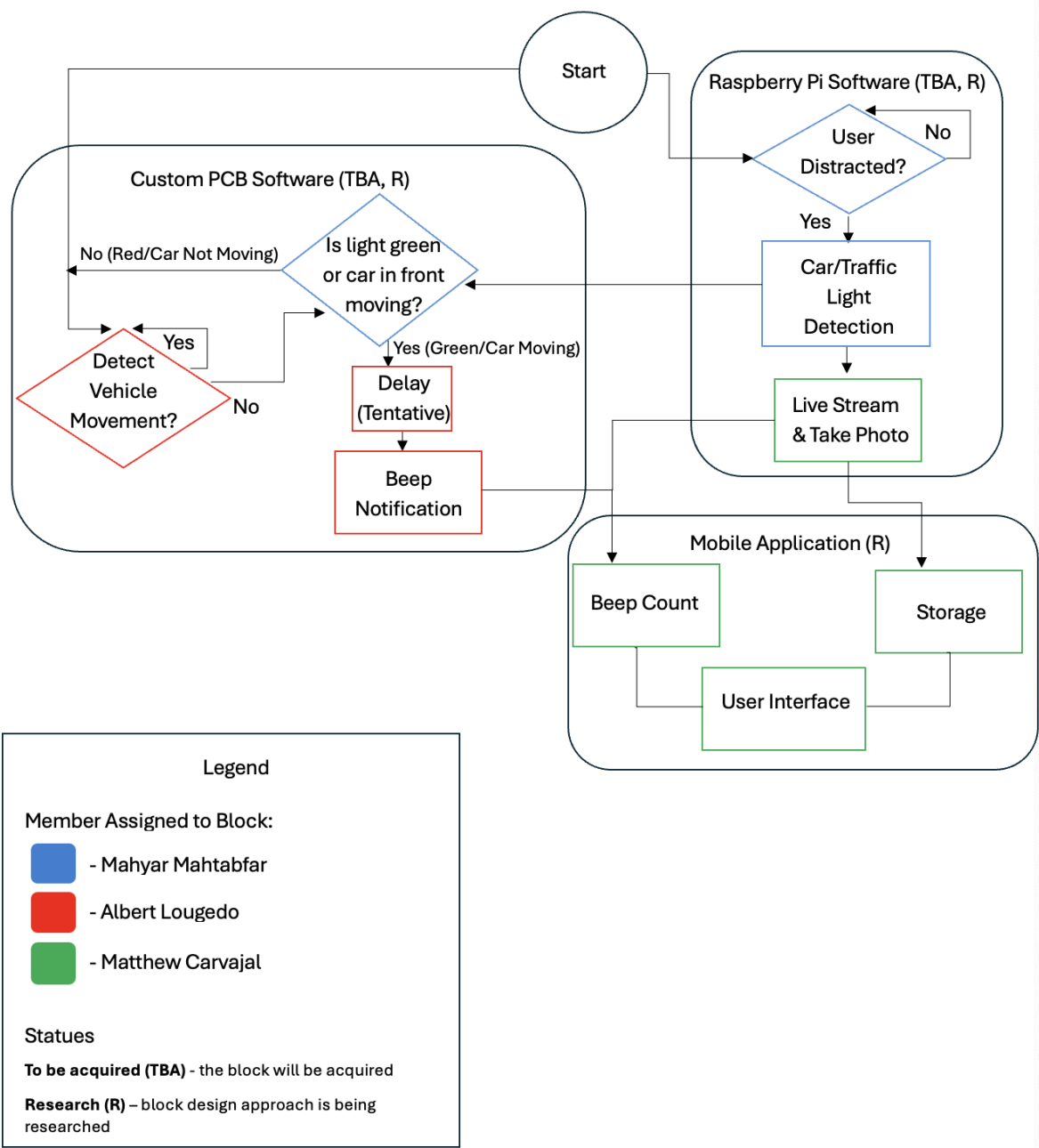
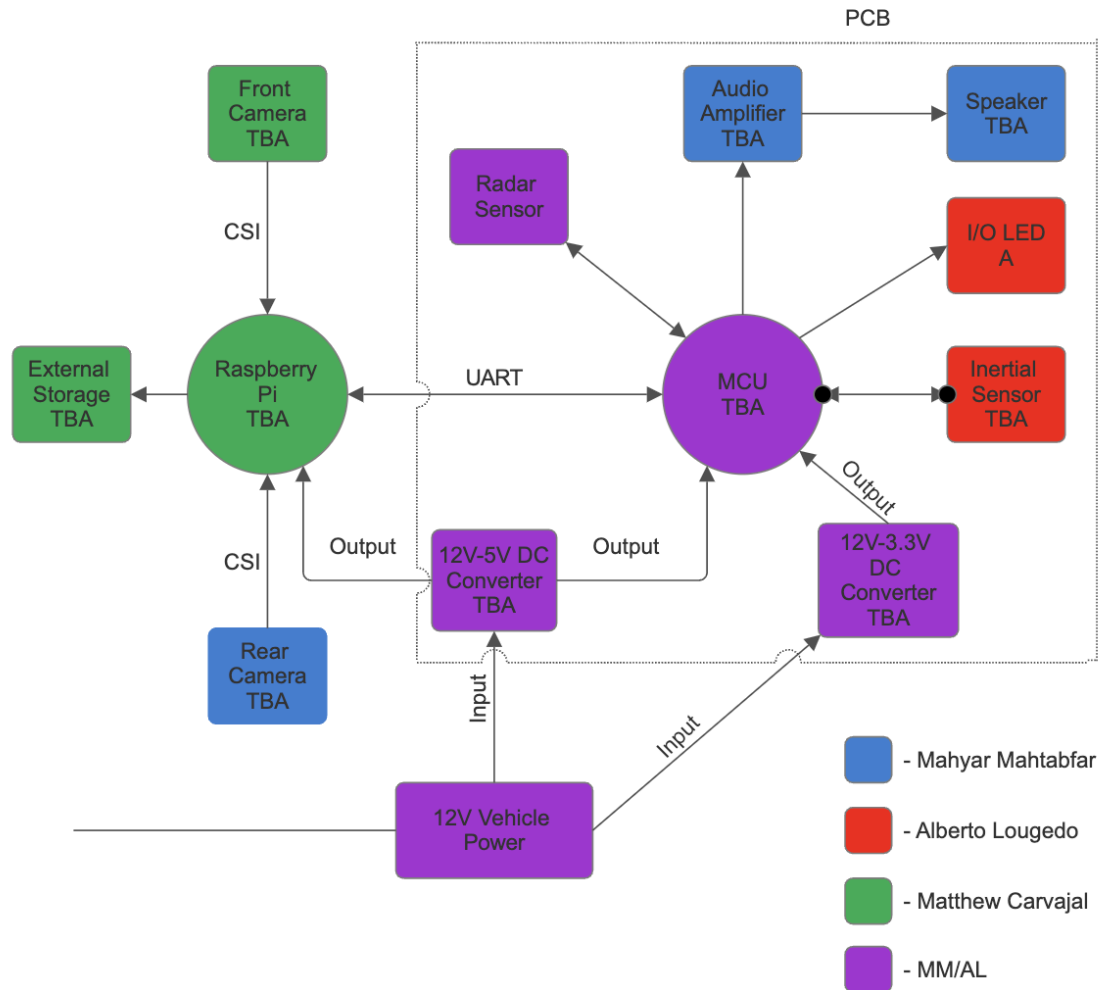
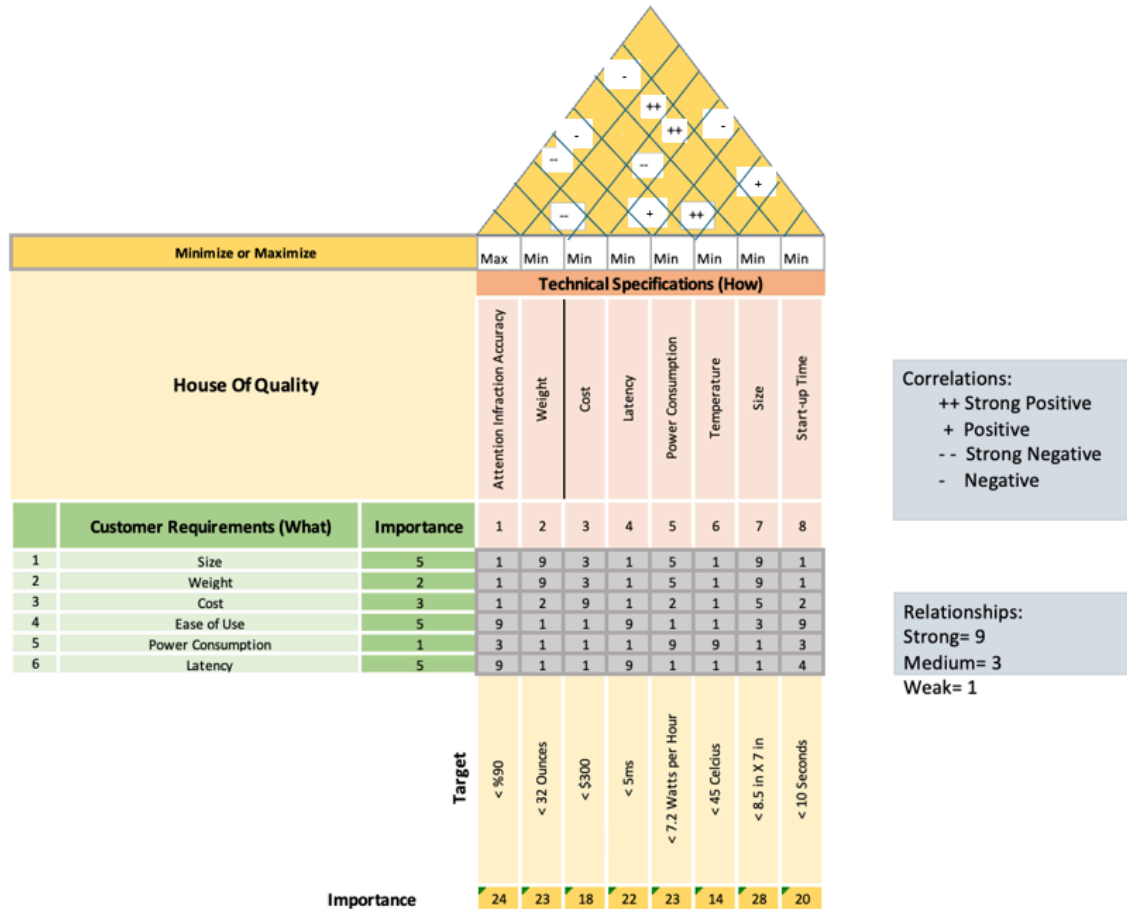


Table 2.5.3 Hardware Block Diagram



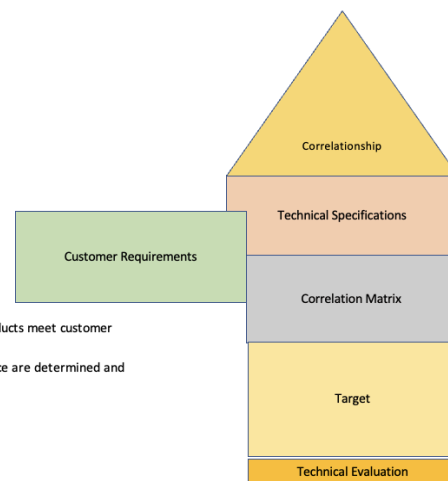
One consideration when designing the project is the needs of the customer. In an effort to take these needs into the design process we have to analyze the tradeoffs of our vision with that of the customer, thus a house of quality visuals was made. In the house of quality we utilized a numerical representation to represent the correlation we deemed fit for the technical specifications and the requirements made by the customer. At the roof of the house is our correlation of the technical specifications to each other. This portion demonstrates the tradeoffs we had to acknowledge outside of the customer's needs.

Figure 2.5.4 House of Quality



Guidelines for using the House of Quality also known as QFD (Quality Function Deployment)

1. Identify Customer Requirements (What)
2. Rate the importance of each requirement, 1 = least important and 5 = most important
3. Identify how the product will satisfy the customer (i.e list Technical Specifications)
4. Identify relationship between technical descriptors and customer need (Strong=9, Medium=3, and low=1)
5. Evaluate which direction the Technical Specifications should go to make the customer happy (max or min)
6. Set up targets for Technical Specifications at the bottom
7. Evaluate competing products or services. The question to be answered here is: How well do competing products meet customer wants? This activity is completely based on research. (1 = worst and 5 = best)
8. Determine the desirable technical attributes. In this step, our performance and the competitor's performance are determined and compared



3. Research

3.1 Technologies

3.1.1 Raspberry Pi

3.1.1.1 Raspberry Pi vs. Jetson Nano

The Raspberry Pi will be utilized because of its processing power, compact and robust design, and affordable price point to run the computer vision tasks in our project. However, how does the Raspberry Pi compare to its competitors? The Raspberry Pi and Jetson Nano are looked upon as competitors when it comes to application in a computer vision project requiring real-time object detection and image processing.

3.1.1.2 Availability and Efficiency

The Raspberry Pi, especially the newer models, is more affordable than the Jetson Nano, consumes less power, comes in a more compact and portable size, and has a better platform for development due to its popularity on a massive global user base. In our case, we will require our computer vision model to detect simple objects and classify between head posture, colors, and contours. These demands do not require the additional parallel computing power from the Jetson Nano which includes an NVIDIA Maxwell GPU with 128 CUDA cores. For intensive deep-learning tasks, the Jetson Nano will perform better when compared to the Raspberry Pi. For our application, since we do not require the extra computing power that makes AI and machine learning possible, provided by the Jetson Nano, and need a smaller more efficient design for application, we decided that the Raspberry Pi is the more ideal fit. When it comes to technology choices to run our required computer vision models in real-time, in a compact efficient design that our product requires, the Raspberry Pi trumps its competitor for computer vision tasks in this application.

3.1.1.3 Hardware Accessories

Other than the Raspberry Pi development board that houses the CPU, memory, and all access ports, there are a few additional accessories we need to consider when meeting the demands of our Safe Sight project. Many projects involving computer vision tasks will drive a high demand on the Raspberry Pi, involving complex calculations on image data that will generate more heat when compared to basic tasks. This calls for one of our accessories to involve a passive or active cooler so that our Raspberry Pi does not overheat. Overheating a Raspberry pi will cause it to throttle its CPU in an effort to lower its overall temperature and heat generation. Furthermore, if this thermal throttling occurs whilst the Raspberry Pi is actively attempting to run computer vision tasks, it will negatively affect performance of the Safe Sight product and inevitably result in the product failing. Additionally, camera accessories and their specifications will be crucial in managing demands on the Raspberry Pi and its heat generation. The goal with both the cameras is to have them capture at speeds and qualities that are efficient enough to prevent thermal throttling and effectively run computer vision tasks in real time.

3.1.1.4 Passive vs. Active Cooling

When it comes to cooling of electronic components there can be two ways: passive or active. Passive cooling utilizes natural conduction, convection, and radiation to cool off components. The use of materials and structures that work well dissipating heat to cool off elements is what passive cooling is all about. For example, passive cooling in small electronics is most commonly seen as a heat sink with thermal tape sticking it to components.

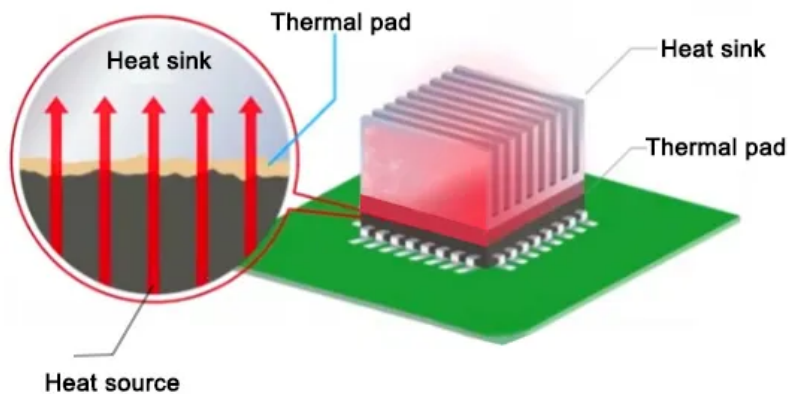


Figure 3.1.1.4.1 Heat Sink

On the other hand, active cooling involves using active components to cool down electronics. The biggest example of this would be a cooling fan that draws power from the electronics its cooling in an effort to dissipate heat through convection. On bigger more demanding electronics, water cooling and pumps might be used as another method of cooling, but for smaller electronics this is not needed.

The method of cooling we have decided to choose to meet our product needs is a mix of both. The Raspberry Pi active cooler uses a combination of thermal tape, metal heat sinks, and active cooling fans to effectively dissipate heat and prevent the thermal throttling effect on its CPU. The Raspberry Pi will begin throttling its CPU at 80°C and increase the effect once it reaches its maximum temperature of 85°C. Considering that our Safe Sight product will be running demanding applications and will be located in a vehicle, we believe that it is only appropriate that we use a combination of the best active and passive cooling technologies to keep our device at optimal running temperatures.



Figure 3.1.1.4.2 Raspberry Pi Active Cooler

3.1.1.5 Camera Accessory

To achieve a product that can process traffic lights and also the drivers posture we will require two cameras. The front road-facing camera will need to operate at a higher resolution than the camera facing our driver. A higher resolution allows for our front road-facing camera to process contours and objects at further distances due to a greater image quality. Since our vehicle position on a traffic light stop most times will not be first, a greater resolution will be required to confirm an accurate interpretation of the traffic light from our computer vision algorithm.

The camera facing the driver will capture video at a much lower resolution since the camera will be a set distance at all times from the driver, making it so the camera can be orientated to have its subject fill in the frame of video capture. Now that the driver makes up most of the frame of capture, a significantly smaller amount of pixels can be utilized to represent their appearance since they are closer to the camera and their posture represents most of the capturing frame. Capturing video this way will alleviate stress from the Raspberry Pi's CPU and allow us to utilize more of its computing power to run the more demanding road-facing computer vision algorithm.

For the driver camera, we have decided to choose the Raspberry Pi Camera Module V2 due to its hardware compatibility and compact size.

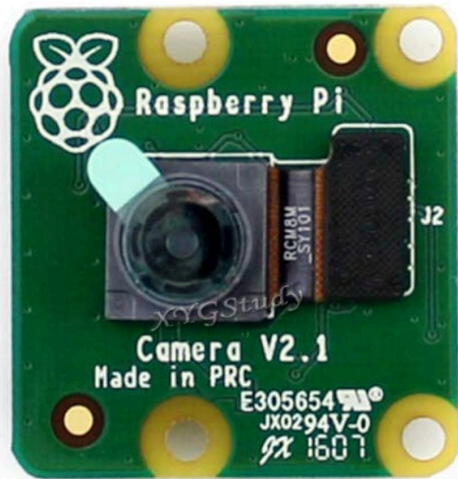
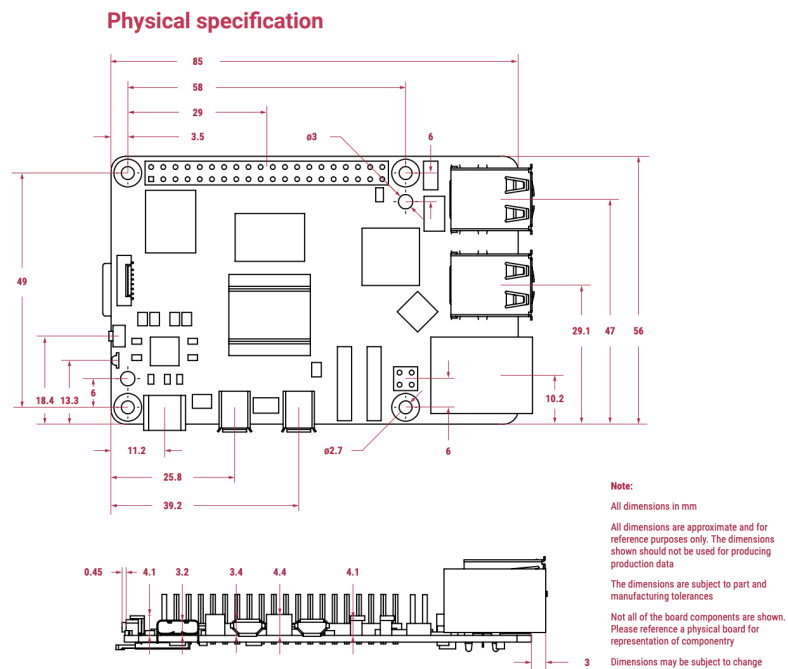


Figure 3.1.1.5.1 Raspberry Pi V2 Module

3.1.1.6 Raspberry Pi 5 Physical Diagram



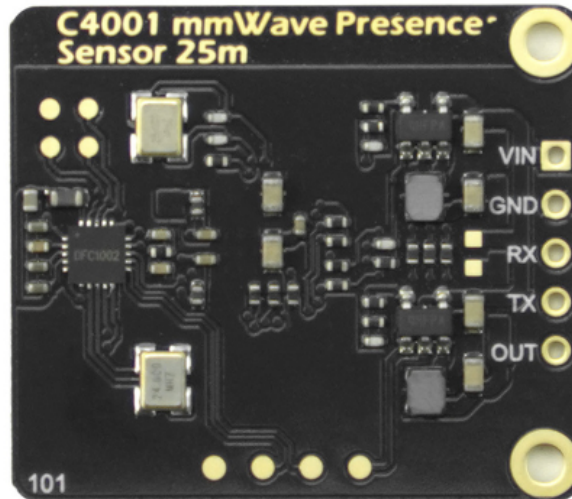
3.1.2 mmWave C4001 24GHz Human Presence Detection Sensor

When looking at the system we are planning on designing, there are many factors we must take into consideration before we make an executive decision on the components at hand. In conjunction with the inertial sensor, which we will be going over in 3.1.3, a radar was suggested by the advisor in order for us to properly address the need for a crash prevention system running off of our PCB.

When conducting our research for the right radar to implement into the board there were many requirements that had to be satisfied. The first major subject was the distance for which the radar can detect moving objects, but this was accompanied with the hurdle of detecting objects through a window. Immediately, the prospect of using an ultrasonic sensor was eliminated. Since the ultrasonic sensor functions by emitting high-frequency sound waves out from the emitter and waits for the waves to bounce back, a windshield will directly send the ultrasonic waves back to the receiver of the sensor. This reflection from the windshield will prevent the waves from reaching any object beyond the glass. The glass in itself has a specific acoustic impedance compared to air, this mismatch in impedance causes most of the sound energy to be reflected at the air-glass interface. This hurdle left us with options such as an optical sensor, Lidar, and camera-based systems. Due to the subsystem of the raspberry pi loading all of the CV conditioning we are throwing at it, the camera based system was not in the picture as we are avoiding the use of a third algorithm for distance detection, thus came into question the Lidar. The Lidar is a sensor that detects light, often working through glass. While a Lidar may seem like the solution to finding the distance of objects ahead of the host vehicle, one big consideration was the reflection/ refraction of light that the windshield glass will reflect on the lasers used from the Lidar. This refraction can cause reduced signal strength while making an effort to reach objects beyond the glass and potential false readings from reflections. Another fault in the Lidar system is that the signal must pass through the windshield twice, once out and another returning. When taking into account robustness of the system we are trying to engineer, having these scenarios of increased margin of error has convinced the research to look at other options. Thus, we have inevitably turned to the radar.

The radar, as a broad technology, serves to identify the presence of objects, determining distance to said object, and thanks to the doppler effect, measures velocity. One functionality we will be utilizing the radar for is collision avoidance, in order to warn the driver using the safesight device of impending impact. Many modern vehicles incorporate radars within their automotive safety system, and these radars have standards we took into consideration when looking for the right radar to implement. Based on the specifications used in the industry, the radars used in these vehicles utilize a frequency band of 77-81 GHz, range capabilities of 1-30 meters, and about 15W for power. These radars utilize advanced signal processing for object detection and these signal processing techniques help with all weather performance which is one thing we have to take into account as well. The radar we have found is the DFRobot C4001 millimeter-wave presence sensor. This sensor is capable of emitting a 24GHz wavelength signal with a range of 25 meters.

Figure 3.1.2.1 DFRobot C4001 Radar

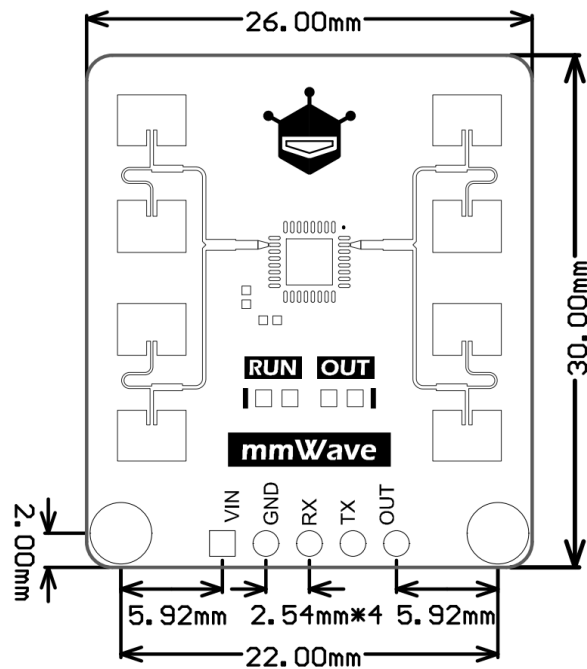


The differences between the mm- wave presence and the infrared sensor are detailed in the following chart:

Figure 3.1.2.2 mm-wave sensor vs Infrared Sensor Comparison

	Millimeter-wave Presence Sensor	Infrared Sensor
Sensing Principle	TOF radar principle + Doppler radar sensing principle (active detection)	Pyroelectric infrared sensing principle (passive radiation)
Motion Sensitivity	Can detect presence, slight movement, and motion of human body	Can only detect motion and close-range slight movement of human body
Sensing Range	Can be adjusted to different sensing distances	Sensing range cannot be adjusted
Environmental Temperature Impact	Not affected by environmental temperature	Sensitivity decreases when temperature is close to human body temperature
Application Environment	Not affected by heat sources, light sources, air flow	Susceptible to heat sources and air flow
Penetration Ability	Can penetrate fabrics, plastics, glass, and other insulating materials	Can only penetrate some transparent plastics
Distance Measurement Support	Yes	No

We decided to compare the functionality as justification for the reasoning of choosing the millimeter wave sensing technology. Since we are implementing this radar for distance detection for a short term alarm with the ambitions to implement a safety crash collision notification, and to sense vehicles moving ahead of the car hosting the device. The 25m distance is more than efficient for our uses paired with a speed measurement range of 0.1 to 10m per second. One other aspect of this radar that makes this system as robust as we would like is that the radar itself is unaffected by snow, haze, temperature, humidity, dust, and light. Ensuring that the radar will be able to withstand weather conditions beyond the windshield.



Embedded in this radar we have chosen is the Frequency Modulated Continuous Wave (FMCW) radar technology. Some notes to mention on this technology are the advantages of simultaneous range and velocity measurement, superior range resolution, and extended detection range. With the simultaneous range and velocity measurement, the advantages of measuring both variables can lead to the implementation of our equations for crash detection and other notifications to the driver of the vehicle. For example, the measured vectors will be labeled as two subcomponents variables to the whole system, where we will introduce feedback in the system in order to measure constant distance recordings of a vehicle in front of the host vehicle. This will then compare the values of our ideal distance, being at maximum range of the radar's constraints, to the actual distance recorded. This feedback will then apply the data to the system where velocity and distance from collision are taken into account. From the FMCW, this can be seamless with simple algebra ringing a notification to the user.

The FMCW's range resolution implements a frequency modulation technique known as the triangular frequency. This modulation pattern involves both the increasing and decreasing frequency linearly, creating a triangular waveform. This separation of the difference frequency from the doppler frequency makes it useful for simultaneous range and velocity measurements to be carried out effectively as compared to the application of an ultrasonic sensor. This modulation technique is crucial for crash detection systems as it provides a comprehensive understanding of the surrounding vehicles' positions and movements. As that is stated, one of the biggest challenges in crash detection is the ability to distinguish doppler shift and frequency shift, and this FMCW technology's inclusion of the triangular modulation enables better discrimination between stationary and moving objects, proving vital for accurately identifying potential collision risks in a dynamic traffic environment. In the talks of the traffic environment there are also a variety of different conditions that require a robust system. With robustness, there are

multiple complex, uncontrollable factors that dictate the system to respond effectively and one of these factors is the ever changing conditions such as weather and lighting. The versatility and integration offered by the FMCW's triangular modulation allows for an increase in reliability in order to overcome these various conditions that are ever so changing, improving the reliability and robustness of the system as a whole. By leveraging these benefits triangular modulation in the FMCW radar significantly enhances the capabilities of crash detection systems in cars, and being able to get this technology to operate on a portable system such as the safesight will be ideal for the system we plan to implement.

Another major aspect as to why we chose this radar as stated before is the distance for which the FMCW is capable of detecting. FMCW radars can utilize large bandwidths on relatively small hardware, allowing the system of safesight to detect preventable accidents in a traffic environment. Outside of the frequency modulation we went in depth for, there are multiple signal processing techniques used within the FMCW that allow for this system such as the beat frequency analysis, fast fourier transform, and the use of advanced algorithms. Beat frequency analysis is a crucial technique used in FMCW radar systems as it is used to determine the range and velocity of targets. This analysis involves measuring the frequency difference between the transmitted and received radar signals, the delta (difference) is the beat frequency and it is directly proportional to the target's distance and relative velocity. The received signal is mixed with a portion of the transmitted signal in the radar's receiver which then produces a signal with a frequency equal to the difference between the transmitted and received frequencies (beat frequency). From obtaining the beat frequency, that signal is then converted to the frequency domain using the Fast Fourier transform (FFT). The FFT efficiently converts the time-domain beat signal into its frequency components as we just stated. This transformation allows the radar to identify multiple targets at different ranges simultaneously. This is accomplished with spectral analysis, which provides a spectrum of frequencies present in the beat signal. Peak detection in the FFT output corresponds to potential targets; these peaks represent the beat frequencies. Threshold application is also applied to the FFT output to distinguish significant peaks from noise, allowing for focused and important data that is not going to alter the performance of the system.

The FFT algorithm efficiently converts the time-domain beat signal into its frequency components, allowing the radar to identify multiple targets at different ranges simultaneously. To detect the threshold, the FFT output is analyzed to find frequency peaks above a certain threshold which correspond to potential targets. The range calculation is the result from this complex process and is how we will be implementing the range calculation into our CV (computer vision) algorithms. The FFT allows for computational efficiency, multiple target detection and high resolution. To achieve multiple targets simultaneously there are many algorithms to implement but we will focus on these two; One algorithm is used here in the FFT processing known as the Cell Averaging Constant False Alarm Rate (CA-CFAR) algorithm. This algorithm enhances detection in complex environments by dynamically adjusting detection thresholds based on local clutter levels, reducing false alarms caused by background variations and calculates a weighted average of clutter data in reference windows. The Multiple Signal Classification (MUSIC) algorithm improves target detection and tracking. This is done by using eigenvalue decomposition of the received signal's covariance matrix and estimating direction of arrival (DOA) of target signals. The MUSIC algorithm in particular is effective in reducing interference from stationary and moving clutter sources.

On the subject of algorithms, the FMCW radar technology can be effectively integrated with computer vision algorithms to enhance object detection, tracking, and classification capabilities. The FMCW radar provides precise range, velocity and angle information. Computer vision on the other hand, offers rich visual features and object recognition. Because of these computer vision algorithms using radar data to improve object localization and tracking, this allows the combination of the two technologies to coexist in an effort to overcome challenging lighting and weather in a dynamically changing environment such as traffic. Mask R-CNN and other similar deep learning models can identify and locate human objects in visual data. The positions of these detected objects are used to guide the FMCW radar in targeting specific areas for vital signal extraction. The versatility of the Convolutional Neural Network (CNNs) can be used to process both the radar and visual data, extracting these complementary features from each modality. The type of algorithm we will be using will be discussed more in the Raspberry Pi section of this documentation, where we will address the CV demands in correlation to the rest of the hardware being used in the system.

Limitations:

In the research for this specific radar technology, there have been many findings that provide a lot of evidence to support the claim that this radar technology is perfect for the use we intend to implement on our system. However, alongside this research came some drawbacks we will have to work to overcome in the talks of this radar. For example, the radar technology we have mentioned contains limited range and doppler resolution, hindering the radar's ability to distinguish closely related objects and detect vulnerable road users, particularly pedestrians. The goal of identifying pedestrians has been a far-reaching goal in itself in order to really execute an effective collision detection system. The main intended use for this radar is to detect bigger, automobile objects in the hopes that the driver will be able to react in a reasonable time. In order to accurately achieve the targeting for these vehicles in traffic and to propagate the signal to the driver, the problem of insufficient resolution must be addressed as well. Even in the higher bandwidths, > 4 GHz, the range resolution may not be sufficient for operating some targets in dense traffic conditions as we see everyday commuting to UCF, stores or experiences. This requires additional separation in the doppler dimension.

Interference with congested environments. In scenarios with many vehicles operating radars in the 76-81GHz band, the power from other radars can significantly exceed the power of echoes from intended targets. This interference can reduce the effective range of the radar to a fraction of its normal capacity. Alongside this interference comes noise. This interference from other radar systems can elevate noise levels, potentially drowning out weak targets as well, meaning that there is a loss on the overall detection rate. This detection interference can lead to ghost targets as well, leading even to false readings. With all of these possibilities in the world of target detection, it is imperative for the team to simulate many different scenarios in an effort to accomplish these hurdles and prove the system to be robust over these challenging circumstances.

These limitations highlight the need for improved radar technologies at the size and thresholds that we are aiming to utilize this specific radar, the mmWave C4001 24GHz Human Presence Detection Sensor. Even though this sensor's technologies and interference mitigation techniques can do the job we are tasking it to do, some modifications can be made in these technologies to enhance the accuracy and reliability of FMCW radar systems in complex traffic environments. For the sake of this project

however, the technologies will be sufficient for the main goal, some alterations will be needed for our stretch goals.

3.1.3 Inertial Sensor

In the scenario where a driver remained idle at a traffic stop and they are not the first driver in line, the system lacked in being efficient in this case. That was until we decided to implement the interital sensor. Prior to this decision being made there were many theoretical approaches to how this problem could be tackled. One of which was implementing a CV algorithm in the hopes that it can determine the distance of the vehicle ahead of us in order to let the driver of the host vehicle know that it is time to go. However effective this may have been, the design lacked complexity and as a result, relied heavily on the subsystem of the Raspberry Pi. So, with that being said the inertial sensor found its purpose. One of the major issues with adding an inertial sensor are the constraints we need in order for the device to be effectively integrated into our system. Firstly, we need a sensor that is ideal for automobile velocity measurement. This is because in order to determine the inertia constant of a vehicle that is driving, there needs to be some sort of temperature threshold and endurance. This is because the dynamically changing environment of traffic can be ever changing from location, time of day, amount of cars around, etc. Being that the environment we are used to is in Florida, this temperature endearment is crucial as we need a sensor that can withstand temperatures of a car as it gets hotter when the car is idle and off. Another constraint is the sensor's ability to be compatible with the ESP32 and can adapt to UART communication to remain consistent with the rest of the hardware in the system as displayed in the hardware diagram earlier in this document. And finally, the last two constraints we demanded to meet were affordability, and compact sizing.

Inertial sensor comparison table:

Feature	MPU-6050	MPU-9250	ICM-20948	LSM6DS3
Manufacturer	InvenSense (now part of TDK)	InvenSense (now part of TDK)	InvenSense (now part of TDK)	STMicroelectro nics
Axes	6 (3-axis accelerometer, 3-axis gyroscope)	9 (3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer)	9 (3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer)	6 (3-axis accelerometer, 3-axis gyroscope)
Gyroscope Range	±250, ±500, ±1000, ±2000 °/s	±250, ±500, ±1000, ±2000 °/s	±250, ±500, ±1000, ±2000 °/s	±125, ±250, ±500, ±1000, ±2000 °/s
Accelerometer Range	±2g, ±4g, ±8g, ±16g	±2g, ±4g, ±8g, ±16g	±2g, ±4g, ±8g, ±16g	±2g, ±4g, ±8g, ±16g
Magnetometer	No	Yes	Yes	No

Digital Motion Processor (DMP)	Yes (6-axis)	Yes (9-axis)	Yes (9-axis)	No
Interface	I ² C	I ² C, SPI	I ² C, SPI	I ² C, SPI
Operating Voltage	2.5V to 3.6V	2.4V to 3.6V	1.71V to 3.6V	1.71V to 3.6V
Package Size	4×4×0.9 mm QFN	3×3×1 mm QFN	3×3×1 mm QFN	3×3×0.9 mm LGA
Status	End of Life (EOL); replaced by ICM-42670-P	Obsolete	Active	Active
Notes	- Integrated 16-bit ADCs- External magnetometer interface	- Integrated 16-bit ADCs- Improved performance over MPU-6050- Integrated magnetometer	- Integrated 16-bit ADCs- Lower power consumption- Enhanced features over MPU-9250	- Integrated 16-bit ADCs- Low power features- Pedometer functions

Many different sensors were able to work for the demands we needed, at one point the team was considering a sensor optimized for the functionality of a smartphone or other portable electronic device. This was a step in the right approach as our system was to be idealized inside of a small component that is transferable from motor vehicle to motor vehicle, but it did not meet the compatibility of UART, cost, or physical constraints for the dynamic environment as we mentioned earlier. Thus we went ahead to find another alternative, the MPU-6050 Accelerometer and Gyroscope. The MPU-6050 Accelerometer and Gyroscope features a 3-axis accelerometer and 3-axis horoscope. These two additives to the hardware are efficient in measuring acceleration, angular velocity, and temperature. The sensor also communicates via I2C, but it can be adapted to UART using additional components. Since the ESP32 is the microcontroller serving the whole system, it is important that our components are compatible with that microcontroller. In the efforts to ensure that the sensor we finalized was going to be ideal, we had to take into consideration physical constraints, power, and introducing other complexities. Now the limitation of the MPU-6050 being the absence of UART functionality and absence of a built-in velocity calculation (resulting in an integration of acceleration data to estimate velocity) caused the choice of the sensor to be this MPU-6050. The main communication protocol being the I2C offloads the UART pins on the ESP32. This is ideal as we have the radar and the Raspberry pi, as aforementioned, already utilizing the UART pins.

3.1.3.1 MPU-6050

The MPU-6050 inertial sensor is an ideal inertial sensor for detecting movement in a device such as the safesight because of key features and advantages over other

sensors. First, the MPU has various degrees of freedom (6) with the integration of a 3-axis accelerometer and a 3-axis gyroscope, enabling it to measure both linear acceleration and angular velocity. This combination provides a comprehensive formulation for motion data, which is what we need in order to track the motion of an automobile efficiently. The sensor's 16-bit ADC for high resolution measurements ensures precision in regard to detection of motion and orientation. This offers configurable sensitivity ranges for both the accelerometer ($\pm 2g$ to $\pm 16g$) and gyroscope ($\pm 250^\circ/s$ to $\pm 2000^\circ/s$), allowing adaptation to varying dynamics of a moving car. The compact size of this sensor is also a very needed constraint that it meets. The dimensions of just 25 x 20 x 7 mm and low power requirements allow the sensor to fit and operate with fatal constraints. Alongside the use of the ESP32 microcontroller we have chosen for this hardware, the MPU-6050 utilizes a I2C communication interface. This is ideal given that we will be communicating to the ESP32 as we will be communicating through the Arduino interface. This allows not only for the software to smoothly communicate but also for the implementation on the hardware as a whole. The ESP32 has only a specific number of pins to communicate with UART, which is already ideal for the Raspberry Pi's integration for the system. The freedom of using I2C allows for such hardware, like the MPU-6050 to be implemented. And finally, the MPU has "shock tolerance" which means it can handle the vibrations of a car moving through traffic (up to 10,000 g), making it robust enough to handle vibrations and impacts in such a dynamic traffic environment.

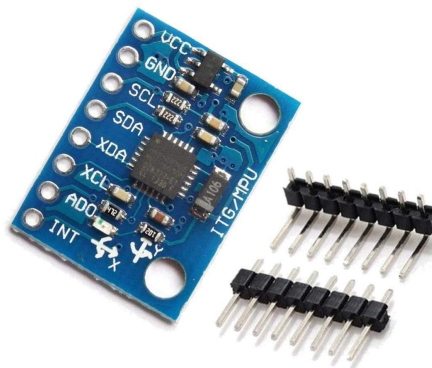
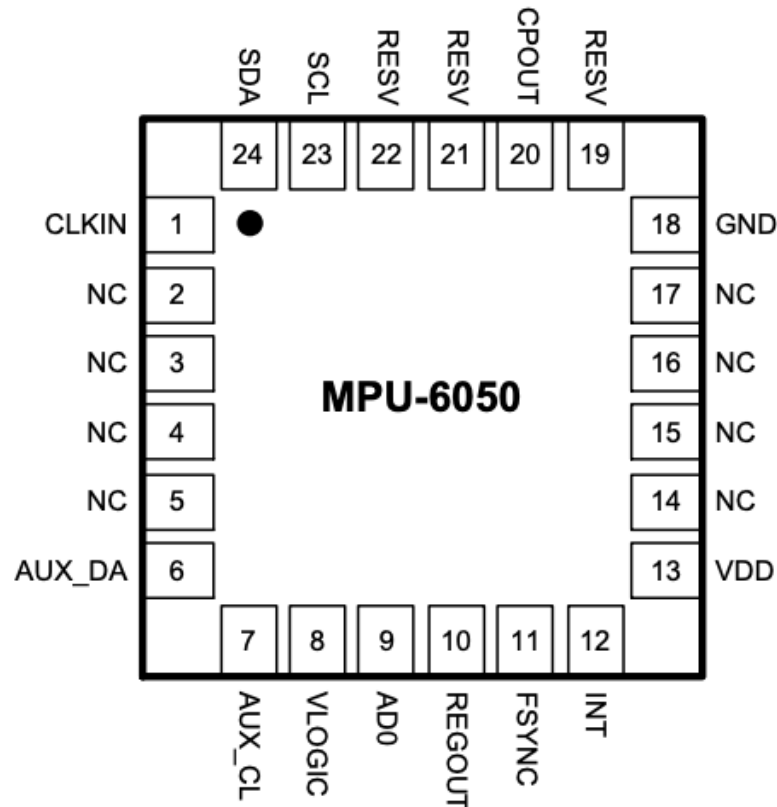


Figure 3.1.3.1.1 MPU-6050

From the visualization provided above, there are specific pinouts shown and hence this section of the document will go in depth on the functionality behind every pin. The VCC and GND pins supply the input voltage and the grounding nodes, respectively. Since this sensor allows for an activation voltage of 3.3V-5V, this makes it ideal for the sensor to be implemented given that the ESP32 provides a 3.3V pin for the rest of the circuit board, allowing utilization of the pin to provide the 3.3V to a node for all 3.3V components. The main connection and pins we will be doing work on are the SCL, SDA, and INT pins. Here in the SCL pin we will be utilizing the I2C communication to provide clock pulses and to establish a connection to the microcontroller's SCL pin. This will enable synchronizing data transfer between the MPU-6050 and the microcontroller. The SDA pin will connect the SDA pins between the sensor and the microcontroller, allowing for bidirectional data exchange between the two components. For the INT (interrupt) pin,

this pin generates an interrupt signal when new sensor data is available. This pin is useful for our application because it allows for the real time data processing without constant polling, allowing the stress of processing power to be brought down.

Top View



MPU-6050 sensor pinout

3.1.3.2 Sensor Limitations

Initially in this section we went over other possible sensors, why they were not a good fit and why the MPU-6050 was the better sensor overall for our application. However, it is also important to note that the MPU-6050 also has its drawbacks. For example, the MPU-6050 has limited accuracy and reliability. The sensor can produce some unreliable readings during rapid or harsh environments. Such changes in pitch or angle may initially result in incorrect outputs before stabilizing which can be problematic in a very intense traffic scene. While we did mention the robustness to shock earlier as a reason for our decision, the sensor struggles with vibrations over 10,000 g. This can lead to erratic readings or clipped values, meaning that we would need to find a way to work around that in order to properly perform data acquisition. On the subject of high G forces, it is also important to note that there is a clipping of the forces when the G force is $\pm 16g$. This can prove to be insufficient for detecting extreme impacts or crashes. This would have to put the responsibility of the crash detection into the CV or the radar to detect when objects have collided with the vehicle in order to implement crash detection.

Working with I2C is a must with the expected usage of the UART pins for the microcontroller being taken, meaning we have a constraint on what sensors we can implement due to the CV module. Being in I2C communication, the data transfer rate is slower than other communication protocols such as SPI-based sensors. This limits its ability to handle high-speed applications requiring faster sampling rates if the environment is too much for the sensor. The I2C communication is not robust over long distances over 1 meter without additional measures like shielding wires and lowering pull-up resistor values, this can lead to complications in the integration for our system if we alter the dimensional constraints.

The MPU-6050 is also prone to noise and requires careful calibration to achieve accurate measurements. Without proper calibration, readings can oscillate and deviate significantly from expected values. This affects the accurate motion tracking which often requires external libraries for sensor fusion algorithms.

3.1.4 Alarm System

3.1.4.1 Speaker and Amplifier

For a sound to be replicated at any time by electronics, a digital sample of the sound signal needs to be stored and then converted, on demand, as an analog signal to then be heard through a speaker. The conversion process from digital signal to analog signal is in need of amplification after conversion. Now, the basis and need for amplification is to take a weak electrical signal and increase its strength, allowing this stronger signal to be sent to the speakers and produce a loud vibration sound to be heard clearly through the speaker cones.

3.1.4.2 ESP 32 Compatibility

The ESP 32 includes two 8-bit DAC channels, which can be used to produce more complex audio output. These DAC channels are hardwired to specific GPIO pins and utilize the reference voltage as output (3.3V). Now, DAC speakers driven by the ESP32 provide more flexibility with sound application. However, the 8-bit resolution provides only 256 discrete voltage levels which is considerably less than CD quality's 16-bit resolution (65,536 voltage levels). This means that distortion will be heard at some levels amongst the signal due to insufficient voltage levels to represent the overall signal. In addition, amplification will be required when utilizing an ESP32's DAC due to its low signal strength not being able to drive most speakers effectively. This additional hardware adds more complexity and significantly increases power consumption. For implementation that requires diverse audio feedback with a range of tones and volumes, DAC driven speakers are perfect for providing this versatility with sound, being able to create alarm sounds, voice instructions, and pleasant tones.

3.1.4.3 Piezo Electric Buzzer

The term "piezoelectric" derives from the Greek word "piezein" which means to "squeeze". This terminology directly reflects the fundamental principles that these devices operate upon. When certain materials are mechanically stressed, they will generate an electric charge. Similarly, when the same materials are energized, they will deform. Piezoelectric material usually consists of ceramics like lead zirconate titanate or

lead magnesium niobate. When these materials receive electrical signals, they transfer the energy into mechanical movement at remarkable efficiency and speed, making them ideal for sound generation and application. Piezoelectric buzzers utilize DC to create a piezoelectric effect that creates vibration sound. The buzzer consists of a multi-resonator, piezoelectric plate, impedance matcher, resonance box, and a housing.

3.1.4.4 ESP32 Compatibility

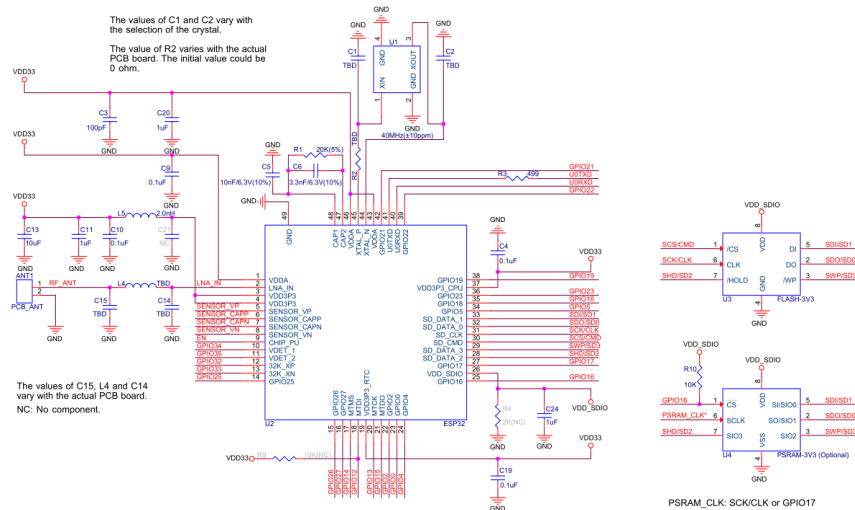
Piezoelectric buzzers are fully compatible with MCUs, especially the ESP32. Coming in passive and active configurations, piezoelectric buzzers are simple in their hardware configuration. Only having two leads and either requiring a simple DC input where the internal oscillator converts this signal to alternating current. Furthermore, the passive configuration requires an oscillating signal at the input provided from the MCU, allowing for different tones to be created instead of just changing voltage for volume in dB. Whether using active buzzers for plug-and-play simplicity or passive buzzers for greater control over sound characteristics, these components offer valuable audio feedback capabilities for a wide range of electronic projects. Understanding the principles, electrical characteristics, and implementation considerations of piezoelectric buzzers enables engineers and hobbyists to effectively incorporate these components into their designs, creating everything from simple notification systems to sophisticated audio interfaces. As electronic devices continue to permeate our daily lives, the humble piezoelectric buzzer remains a critical component for bridging the gap between digital systems and human perception.

3.1.5 MCU

When it comes to picking the right microcontroller, there are many things that need to be considered. Questions arise such as: “What is the price of the MCU?”, “How much power does it consume?” or “Is this MCU even compatible with my system?”. These are all very important questions that must be answered when picking the right microcontroller for your project. Below, some, if not, all of the questions will be answered so that this project can have the perfect MCU that will accomplish all of our tasks.

3.1.5.1 ESP32

The ESP32 is a popular and versatile system-on-chip microcontroller that offers a range of features suitable for many different IoT and embedded systems projects that was developed by Espressif Systems in September 2016. This MCU contains a Tensilica Xtensa LX6 32-bit dual-core processor with a clock speed up to 240 MHz. This processor can provide enough processing power for the most demanding applications. In regards to memory, it holds 4 MB of flash memory with 520 KB of RAM. It has the most amount of memory compared to the rest of the microcontrollers that are going to be discussed. This will allow it to support complex tasks with large amounts of data. One of its key advantages is built-in Wi-Fi and Bluetooth connectivity, enabling seamless wireless communication for IoT applications. Additionally, the ESP32 has a wide variety of GPIO pins, analog-to-digital converters, digital-to-analog converters, pulse-width modulation, I2C, SPI, UART, and capacitive touch sensors, making it highly adaptable for different types of projects.



ESP-32 Schematic for applicational use

Thanks to its high processing power, wireless connectivity, and wonderful energy efficiency, the ESP32 is well suited for this project due to its ability to enter low power modes and being able to operate on battery power for extended periods, making it an excellent choice for remote monitoring systems such as Safe Sight. The ESP32 is compatible with a variety of operating systems and development platforms. These platforms include Arduino IDE since it is probably the most widely supported MCU in the Arduino community. It allows users to program in the common Arduino C++ framework with the help of the ESP32 Arduino Core. It can also handle being developed on Linux, MacOS, and Windows operating systems as well. Making it the perfect MCU for all developers, no matter their choice of operating system. The creator of the ESP32, Espressif Systems, has their own software called Espressif IDF. It is the official development framework from Espressif, offering low level control and optimization for advanced projects. It is best suited for developers who need to build firmware from scratch while having full control over the hardware interactions. Therefore, the combination of affordability, robust performance, capability, and having such a strong developer community, makes the ESP32 a go-to decision for most projects.

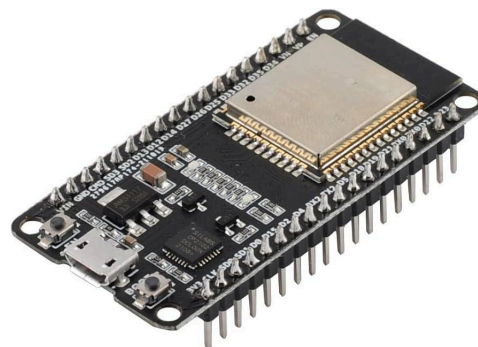


Figure 3.1.5.1.1 ESP32 Microcontroller

3.1.5.2 Raspberry Pi Pico

The Raspberry Pi Pico is a versatile and low-cost microcontroller board developed by Raspberry Pi in January 2021. It is Raspberry Pi's first board based upon a single microcontroller chip. It has a dual-core ARM RP2040 processor that runs up to 133 MHz with 264 KB of RAM and 2 MB of on-board flash memory. You can imagine the Pico as a smaller version of the Raspberry Pi, even though it is similar to the ESP32, as it implements much of the same features as the ESP32 including WiFi and Bluetooth connectivity. However, it does have a different architecture and development ecosystem than the ESP32. Unlike traditional Raspberry Pi boards, which can run full operating systems, the Pico is designed more for embedded systems applications and runs bare-metal code with lightweight runtime environments such as MicroPython and CircuitPython.

With the price range of \$4-\$10, the Raspberry Pi Pico comes with a Programmable I/O (PIO) system, which allows for efficient handling of custom I/O protocols without putting too much strain on the CPU. It has 26 multi-function GPIO Pins, including 3 analog inputs that support interfaces such as I2C, SPI, UART, PWM, and ADC, making it one of the most ideal MCU's for hobbyists. The Raspberry Pi Pico has been optimized for low power consumption as well, making it a good option for this project due to low power consumption. It supports an input voltage of 1.8 - 5.5 V of DC power and can operate in temperatures from -20°C - 85°C, making it a perfect microcontroller for all conditions.

Multiple programming languages are supported on the Raspberry Pi Pico. Languages such as MicroPython, CircuitPython, C/C++, etc. For beginners, MicroPython is a popular choice due to its simplicity. With MicroPython, you can develop in a high-level programming environment, while writing simple, easy to learn code. Unlike C/C++, MicroPython doesn't require setting up a development environment and a compiler, MicroPython allows users to write and execute code interactively using the REPL (Read-Eval-Print Loop) interface using a serial connection. Beginners and advanced developers can develop code super efficiently because there is no need for recompilation. MicroPython also has straightforward syntax for controlling multiple interfaces (GPIO, I2C, etc.) with built-in support for plenty of peripherals, reducing the complexity of embedded programming. On the other hand, Circuit Python is a beginner-friendly version of MicroPython that was designed to simplify programming microcontrollers such as the Raspberry Pi Pico. One of its most popular features is its drag-and-drop programming model. This allows users to plug up a MCU to the computer, view it as a USB storage device, and edit their code directly in a text editor, without needing to compile or run any flash tools. Like MicroPython, CircuitPython includes built-in libraries and modules that make working with peripherals a breeze. Finally, developing in C/C++ on a Raspberry Pi Pico offers immense control, efficiency, and performance compared to high-level languages like MicroPython or CircuitPython. The Pico's official development environment is C/C++ for the SDK. This provides a long list of libraries that are able to interact with the Pico's different interfaces. Developers use C/C++ for highly optimized, low-latency applications, which is great for real-time processing. C/C++ is also very versatile as it can be compiled on various RP2040-based boards, without the need for modification. This is extremely useful in case the Raspberry Pi Pico isn't the right board for your project. You can easily move to another RP2040-based board with more or less features and save time writing new code. Although C/C++ has a stronger learning curve than Python based languages, it is often the best choice for projects that prioritize efficiency and deep hardware control. Therefore, developers love the Raspberry Pi Pico due to its versatility, features, cost and power consumption.

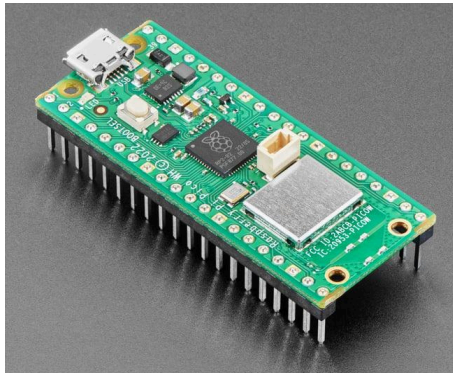


Figure 3.1.5.2.1 Raspberry Pi Pico Microcontroller

3.1.5.3 STM32 Series

The STM32 microcontrollers have been developers' go-to choice for various electronic projects for a while. The family of microcontrollers was developed by STMicroelectronics in 2007. They are based on ARM Cortex-M processors that can range from 24 MHz to 480 MHz. Each MCU from STMicroelectronics consists of ARM processor cores, flash memory, static RAM, debugging interfaces and various peripherals. In particular, the STM32H742 MCU features the highest clock speed and memory capacity out of all of the STM32 series. With a clock speed of 480 MHz and 512 KB of RAM with 2 MB of flash memory, this MCU is a beast for most embedded systems and electronics projects. If you are looking for a microcontroller with extreme low-power consumption, the MCUs with the Cortex-M0/M0+ are perfect for applications that focus on saving power (STM32L series). Some of the MCUs can operate as low as 1.7 Volts which is perfect for devices that need to be active for long periods of time. If you are looking for super high performance out of an MCU, you can look for models with the Cortex-M4/M7. These models offer high performance with DSP (Digital Signal Processing) and FPU (Floating-Point Unit) capabilities. Depending on your needs, the STM32 series microcontrollers can run anywhere from \$5-\$30, making it one of the more pricier options on this list.

Even though there is a higher price point for the STM32 series microcontrollers, they do come with an abundance of peripherals, including GPIO Pins, ADC, DAC, PWM, UART, SPI, I2C, CAN, USB, and Ethernet. Many models also have hardware accelerators for cryptographic functions, making it perfect for applications that need security. Some high-end STM32 chips even have graphics acceleration for display based applications. Like the ESP32, there are even some wireless STM32 options that have Bluetooth as well, making it suitable for this project. The STM32 series microcontrollers can be used in a variety of development environments too. These environments include STM32CubeIDE, which is STMicroelectronics official IDE for the STM32 series microcontrollers. Some businesses on the professional side, use Keil MDK and IAR Embedded Workbench, which are perfect for embedded system development. Some hobbyists use Arduino IDE for some STM32 programming because it is user friendly and compatible with a variety of other microcontrollers. Like the Raspberry Pi Pico, a select number of STM32 series microcontrollers can be programmed with MicroPython for efficient development. Therefore, the STM32 microcontrollers offer high flexibility,

performance and extreme power efficiency, making them a go-to choice for beginners, hobbyists, and advanced developers alike.

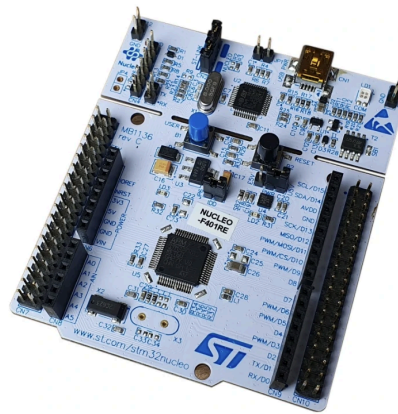


Figure 3.1.5.3.1 STM32 Nucleo F411RE Microcontroller

3.1.5.4 Arduino Nano

The Arduino Nano is a compact and versatile microcontroller that was released in 2008 by Arduino. It is based on the ATmega328P processor and offers similar functionality to the Arduino Uno, but in a more compact form factor. It features a 16 MHz clock speed with 2 KB of RAM and 32 KB of flash memory. With an operation voltage of 5 V, it has a wider compatibility with most sensors and modules compared to other microcontrollers. Many of the older sensors, LCDs, and modules (ultrasonic sensors, character displays, etc.) are designed for 5 V operation which will eliminate the need for modifying the voltage via level switching when communicating with these interfaces. The Nano is probably one of the most breadboard friendly boards out there today. This allows prototyping of components to be super easy. Strictly because most prototyping components are designed with 5 V in mind. This will make integration of components easier without the need to voltage regulation during the prototyping process. The Nano is also one of hobbyists most recommended MCUs right next the ESP32 due to its capability to power more powerful peripherals. This is because with a higher operating voltage, it can output more current per pin, which other lower-voltage microcontrollers can't do.

Since the Arduino Nano has one of the best operating voltages, that means that it can have more pins. The Nano offers 14 digital I/O pins (input/output), which can support 6 PWM and 8 analog input pins. Unlike the Arduino Uno, the Nano relies on USB power (via USB Mini-B) rather than a DC power jack. The Nano can be programmed in Arduino IDE with the same code as other ATmega328P-based boards. The only downside to the Arduino Nano is that it only supports one UART, one I2C, and one SPI interface. This is due to its compact size and microcontroller limitations. Compared to other MCUs like the ESP32 and the STM32, where they support multiple UARTs, I2C, and SPI buses. However, there are ways to expand its communication capabilities. You can use a SoftwareSerial that allows for additional UART communication using digital pins. This method is slower and less reliable than traditional hardware UART. For I2C, you can use I2C multiplexers that enable multiple I2C buses on a single Nano. For SPI, you can use

SPI Chip Select Management which allows for SPI devices to share the SPI hardware bus by using different chip select pins, allowing for multiple devices on one SPI interface.

Overall, the Arduino Nano has been deemed one of the best by hobbyists and developers alike. This is due to its power, compact size, and user friendliness. It has the same performance as the Arduino Uno with the processor and memory specs, just at a smaller size. With the cost anywhere between \$5-\$20 dollars, it makes it one of the average priced MCUs for small form factor projects. Some companies have even cloned the Nano to make it even cheaper. While all of this sounds great, the Nano does lack some I/O pins strictly due to its size but, if you need a MCU that is reliable, tiny, and overall very versatile, the Arduino Nano is for you.

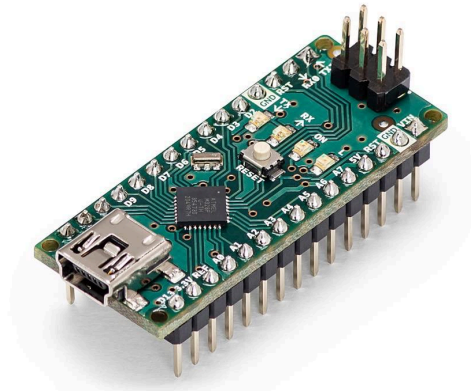


Figure 3.1.5.4.1 Arduino Nano Microcontroller

3.1.5.5 Teensy

The Teensy is a powerful and compact microcontroller development board that has gained massive popularity among hobbyists and embedded system developers. It was designed by PJRC in 2008 when the Teensy 1.0 microcontroller was released. The recent Teensy family (Teensy 4.1) features a ARM Cortex-M7 processor running at a clock speed of 600 MHz. It also has 1 MB of RAM with 8 MB of flash memory, making it suitable for a wide range of applications. The Teensy 4.1 features additional features like Ethernet and microSD card support as well. Don't let its compact size fool you, this family of MCUs has many capabilities, with numerous GPIO pins and support for various interfaces including UART, SPI, I2C, CAN, and USB.

One of the Teensy's key strengths is its compatibility with the Arduino IDE through the Teensyduino add-on feature. This feature allows for easy and seamless programming in C/C++. It also operates at a voltage ranging from 3.3 V - 5 V, making it super compatible with a wide range of sensors and peripherals. The Teensy is flexible enough to be used in high-performance applications requiring maximum clock speed, to low-power scenarios where conserving energy is top priority. This accessibility, combined with its high performance, make the Teensy a wonderful choice for projects that involve real-time systems. Its vibrant community and extensive documentation resources make the Teensy super appealing to developers, providing ample resources for beginners and experienced developers to explore its full potential when it comes to embedded systems.

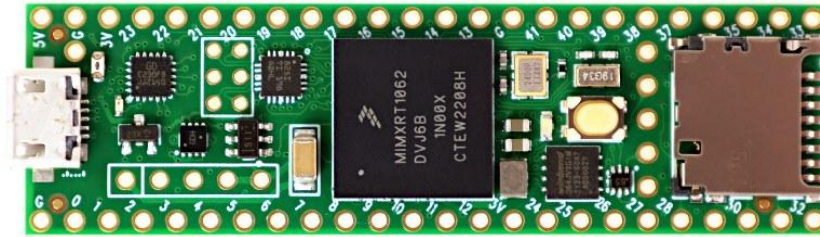


Figure 3.1.5.5.1 Teensy 4.1 Microcontroller

Board	Processor	Clock Speed (Max)	Flash Memory	RAM	Communication Interfaces	Operating Voltage	Price
ESP32	Tensilica LX6	240 MHz	4 MB	520 KB	I2C, SPI, UART, Wi-Fi, Bluetooth	2.2 - 3.6 V	\$3 - \$10
Raspberry Pi Pico	RP2040 (ARM Cortex-M0)	133 MHz	2 MB	264 KB	I2C, SPI, UART, WiFi, Bluetooth	1.8 - 5.5 V	\$4 - \$10
STM32	ARM Cortex-M	480 MHz	2MB	512 KB	SPI, I2C, UART, USB, Ethernet, CAN	1.7 - 3.6 V	\$5 - \$30
Arduino Nano	ATmega328P	16 MHz	32 KB	2 KB	I2C, SPI, UART	5 V	\$5 - \$20
Teensy	ARM Cortex-M7	600 MHz	8 MB	1 MB	I2C, SPI, UART	3.3 - 5 V	\$20 - \$35

Figure 3.1.5.5.1 MCU Comparison Chart

3.1.5.6 Our Selection: ESP32

With all of the Microcontrollers that we compared, our group ended up going with the ESP32 for our microcontroller of choice. This was because of a multitude of reasons. The main reason was because it is one of the only MCUs that we researched besides the Raspberry Pi Pico that offered bluetooth. Bluetooth was essential for us because we need to connect our Safe Sight device to our app in order to view the data that it is going to capture. Also, it does offer 520 KB of RAM which is more than capable to handle our Raspberry Pi that will be our secondary system for computer vision. Even though the speed on the ESP32 isn't the highest out of all of the other MCUs that were researched, 240 MHz will be perfect for processing the data needed for the sensors and the information coming from the Raspberry Pi since all the ESP32 is going to handle is our beep sensor and the radar that is going to be installed on the PCB. Therefore, with budget in mind, we decided to go with the ESP32 for its overall power and compatibility with a majority of the other systems in our project.

3.1.6 MCU Communication Protocols For Components

3.1.6.1 UART Communication Protocol

UART stands for Universal Asynchronous Receiver and Transmitter. UART communication utilizes a serial approach, sending data bits one at a time over a wire.

This protocol is asynchronous, meaning that each device relies on its own internalized clock speed and configuration to maintain the same baud rate or data transmission speed for successful communication. All devices that are compatible with UART communication will have pinouts labeled TX and RX, standing for transmitting and receiving. These pins will be cross connected to each device meaning that the RX node on one device will lead to the TX node on another. Additionally, UART communication in embedded systems is packaged into frames for efficient transmission.

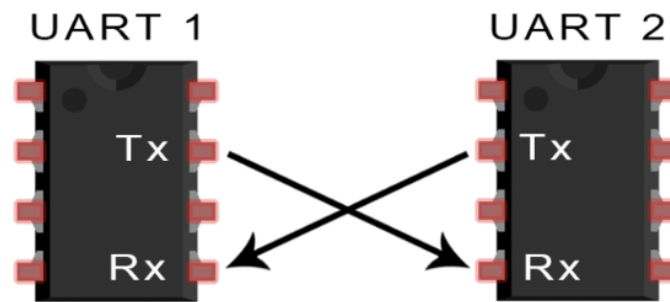


Figure 3.1.6.1.1 UART Configuration

Now the size of a serial frame being interpreted by a device in communication can vary (5-8 data bits or 1-2 parity bits). However, each and every frame being sent or received includes a starting bit, data bits, parity bits, and stop bits. Start bits are first received and are always 0, they signify the start of incoming data bits and represent a low-voltage wakeup call for the receiving device that a transmission is incoming. Furthermore, data bits in a frame are received in the order from least significant bit first to most significant bit last. Depending on the n amount of bits in the data transmission you will have 2^n amount of unique transferable characters. Leading after the most significant bit, the parity bit is set to either check for even or odd parity based on the amount of 1s in the data transmission. This parity bit is compared with the receiving device's own parity calculation bit when reading the data bits and is able to make a decision on whether or not there is a mismatch error. If the parity bit in transmission does not match the receiving device's parity bit from deciphering the frame, then a conclusion can be made that there is a transmission error. Finally, the last bit in the UART communication protocol frame is the stop bit. Opposite from the start bit, the stop bit represents the end of transmission and is always a high-voltage signal. The stop bit is 1 and following its interpretation, the receiver can prepare for the next incoming frame.

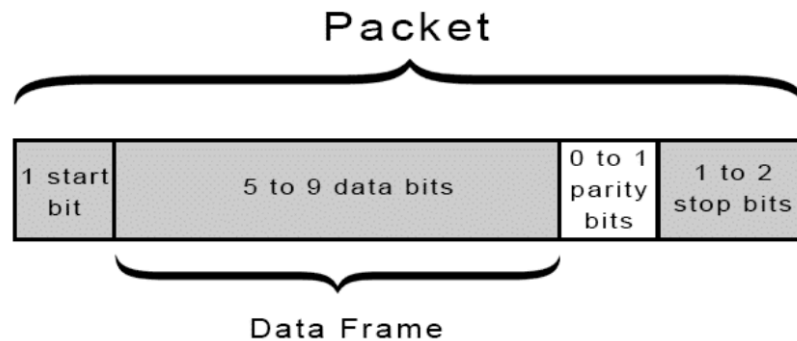


Figure 3.1.6.1.2 UART Frame

When compared to I2C and SPI, UART is slower due to its asynchronous nature but makes up for this in its universal implementation. Overall, UART communication protocol is a popular and sought after choice in utilization in embedded systems applications. Its asynchronous protocol and simple wiring design for communication provides much needed efficiency and accessibility when debugging and implementing new components and systems.

3.1.6.2 SPI Communication Protocol

SPI stands for Serial Peripheral Interface and is a synchronous type of data communication, meaning both devices are coordinating their clock speeds to run at the same transmission rate. Additionally, SPI is beneficial when compared to I2C or UART because it allows for data to be transmitted without interruption. Instead of needing to be packaged into frames and transmitted, data bits in SPI protocol can be transmitted and received in a continuous stream. The hardware consists of four different nodes for communication making it more complicated to configure. However, each node is vital in its contribution to the system.

A system communicating using SPI will have a master and slave dynamic. The master device will serve as the controlling center in this communication protocol and communicate with its individual or multiple slave devices. SPI allows for multiple devices to be connected and communicate with one another and provides a platform for complex systems to operate as one. In its configuration, SPI consists of MOSI, MISO, SCLK, and SS/CS nodes. MOSI and MISO lines of communication are for the master device to send data to the slave (Master Out Slave In) and for the slave to send data back to the master (Master In Slave Out). SCLK represents the wire for the serial clock synchronization, and SS/CS is the master select line where the master selects which slave to send data to in the case of multiple slaves.

The clock signal for this type of synchronous communication is generated by the master device and is connected to each slave to operate upon. The clock signal in SPI can be modified by using the properties of clock polarity and phase. For example, clock polarity can be set by the master to allow for communication bit signals to be sampled or transmitted with the rising or falling edge of the clock cycle. If the system using SPI has

multiple slaves, they will all share the same clock node wired in parallel and have individual nodes for SS/CS to be individually accessed from the master.

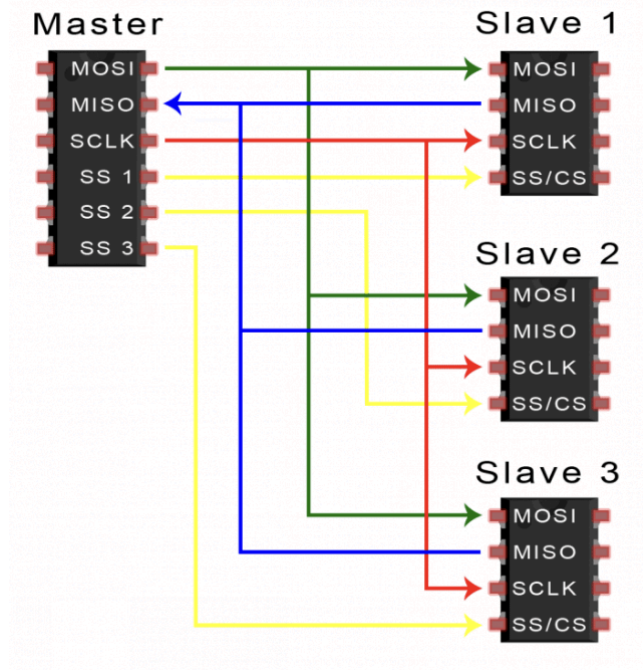


Figure 3.1.6.2.1 SPI Configuration

Disadvantages of SPI include its complicated implementation and hardware setup, absence of a method for error checking (parity bits), and configuration only allowing a single master. However, advantages of SPI include its exceptionally high speed, full-duplex communication, and no need for start and stop bits.

3.1.6.3 I2C Communication Protocol

I2C protocol (pronounced I squared C) stands for inter-integrated circuit, it is a synchronous protocol which allows for multiple slaves connected to a single master (similar to SPI) or multiple masters connected to one or multiple slaves. Similar to UART, I2C sends its data in messages that are then broken down into frames of data. Each message has an address frame that contains the binary address of the sending device and a frame containing the actual data. Furthermore, the message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame.

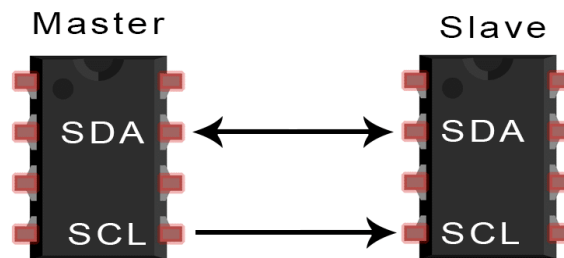


Figure 3.1.6.3.1 I2C Configuration

A scenario where a message would be sent from a master to a slave would play out as such. The master will send a message to another device connected to the SDA line by first setting the SDA line from high to low voltage and as the start condition for the message to be sent. Then, the address frame will be sent to single out the specific device that wants to be communicated with followed by a read/write bit. A read bit sent means the master will want to pull data from the receiver, and a write bit signifies that data is being sent to the receiver. After the address frame including the read/write bit is received by the receiving device, an ACK/NACK bit needs to be sent back to the master to signify that the data frame was received correctly. Now that a connection has been established between the two, data has to be sent in 8 bit frames followed by ACK/NACK bits for confirmation. Finally, when the exchange comes to an end between the two devices, the stop condition signifies the SDA line changing from low to high voltage.

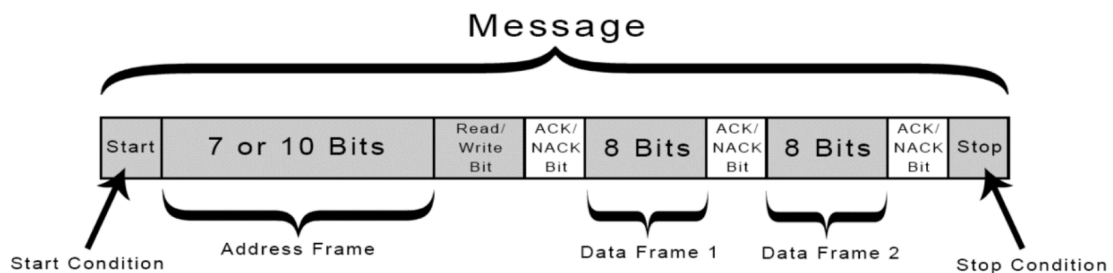


Figure 3.1.6.3.2 I2C Frame

I2C is widely used in embedded systems for its simple hardware configuration and welcoming implementation when working with multiple devices at once. Only requiring two wires and its ability to support multiple slaves and masters in one system makes it a strong choice in embedded systems projects. Hurdles with using I2C communication protocol will arise when the message being sent exceeds 8 bits and requires quick processing/speed. However, I2C is one of the most popular methods for synchronous communication amongst electronic devices.

3.1.6.4 Summary

All of the common embedded systems communication protocols discussed in this section are excellent choices, each excelling in specific applications depending on the project's requirements.

For example, UART (Universal Asynchronous Receiver-Transmitter) is a straightforward and reliable option when an application demands basic communication with minimal setup and quick execution. It is particularly well-suited for simple, point-to-point data transfer scenarios. On the other hand, SPI (Serial Peripheral Interface) surpasses UART in situations where ultra-fast data transfer between two devices is critical throughout the application. Its high-speed capabilities make it a preferred choice for projects requiring rapid and continuous data exchange. Finally, I2C (Inter-Integrated Circuit) stands out as the ideal protocol when an application involves connecting multiple devices on the same data line bus to communicate synchronously. This makes it highly effective for systems with numerous peripherals that need to share a common communication channel efficiently. Each protocol has its strengths, and selecting the right one depends on the specific needs and constraints of the project at hand.

	UART	I2C	SPI
Full Name	Universal Asynchronous Receiver/Transmitter	Inter-Integrated Circuit	Serial Peripheral Interface
Communication Type	Asynchronous	Synchronous (Multi-Master, Multi-Slave)	Synchronous (Full-Duplex)
Number of Wires	2 (TX, RX)	2 (SDA, SCL)	3+ (MOSI, MISO, SCK, CS)
Clock Signal	No	Yes	Yes
Device Identification	None	7-bit or 10-bit addresses	Through Chip Select (CS) lines
Data Transfer Speed	Configurable baud rate	Typically slower than SPI, medium data rate	Faster, suitable for high-speed data transfer
Use Cases	General-purpose serial communication, connecting devices to microcontrollers.	Connecting multiple devices to a single bus, sensors, EEPROMs, etc.	High-speed data transfer between microcontrollers and peripherals such as displays, SD cards, etc.

Figure 3.1.6.4.1 Communication Chart

3.1.7 Development Environment: Languages and Repositories

3.1.7.1 C

C is a general purpose programming language developed by Dennis Ritchie in the early 1970's. It was originally created for constructing utilities running on Unix and later used to re-implement the Unix operating system kernel. C has since become one of the most widely used programming languages due to its efficiency, portability, and versatility. One

of the main features of C is that it has low-level access to your machine. C provides direct access to memory and hardware, making it suitable for low-level programming at the system level. Static typing is also mandatory in C where the variables must be declared before use, with their respective types stated. C has often been described as a low-level language just because it is very easy and simple to use. Unlike true low-level languages like assembly, C offers a higher level of abstraction, making it more of a “middle-level” language in some contexts. Because of this, C does need to be compiled in order to be translated to assembly and then machine code in order to finally be executed and run the program. Compared to other languages like Java, C does not have many object oriented components involved in it. C is rather used more for hardware development, compared to Java, especially in embedded systems and semiconductor programming. Over time, C has significantly influenced many modern programming languages, including C++, Java, and C#. Its syntax and concepts serve as a foundation for other programming languages which is why some people refer to it as the “mother of programming languages”.

3.1.7.2 Swift & Swift UI

Swift is a modern, general purpose programming language that was developed by Apple Inc. in 2010 for iOS, macOS, watchOS, and tvOS app development. It was first introduced in 2014 and has since become a primary language for Apple platform development. Swift is known for its safety. It was designed with a feature called optionals, this feature handles null and undefined values without the risk of runtime crashes. This was designed in order to eliminate entire classes of unsafe code therefore, reducing risk. Swift offers high performance compared to other C-based languages. Apple claims that search algorithms in Swift can be up to 2.6 times faster than Objective-C and up to 8.7 times faster than Python 2.7. Swift also uses a very modern syntax that results in very clean and well organized code. This allows a programmer to write less code to perform the same tasks as other languages. Compared to C, Swift uses Automatic Reference Counting (ARC) to handle memory allocation and deallocation automatically. ARC is a memory management mechanism that automatically tracks and manages the app’s memory usage. It ensures that objects are kept in memory as long as they’re needed and released once they are no longer in use. ARC has a reference counting style. Every instance of a class keeps track of how many strong references point to it. When a new strong reference is made, the reference count increases, and when a reference is removed, it is decremented. When the reference count hits 0, then the object is automatically deallocated, freeing up memory unlike many other languages where you have to use free() to deallocate memory manually. Since 2015, Swift has become open-source, which has gained a massive community and ecosystem of developers all around the world.

Swift UI is a user interface toolkit introduced by Apple in 2019. It allows developers to design and develop user interfaces for all apple platforms using Swift. Swift UI uses Swift as its main programming language. While SwiftUI focuses on the UI construction, Swift is used for all aspects of app development, including logic, data handling, and backend operations. This toolkit brings over Swift’s declarative syntax. This allows developers to state what the user interface should do, making the code more intuitive and easier to read. SwiftUI offers a real-time preview of your interface as you code, so you don’t have to build and compile your code each time you want to test, therefore, making the development process more efficient. With all of the different features Apple offers to their clients, SwiftUI has support for Dark & Light Mode and Localization just by a simple

toggle. Reducing the amount of code that developers need to write. The only downside between Swift and SwiftUI is compatibility. Since these frameworks are a part of the Apple ecosystem, when it comes to Android development, there is little to no compatibility for application development, the developer would need to seek other options to develop for all platforms. However, since this project is a prototype, we can focus on developing for the masses later on after our prototype is complete. Therefore, these two platforms together provide a powerful toolset for developing applications across Apple's ecosystem, offering improved safety, performance, and developer productivity.

3.1.7.3 Java

Java is a versatile, object-oriented programming language developed by James Gosling in 1995. It is both a programming language and a software platform that has become one of the most popular development platforms worldwide. Due to Java's portability and versatility, it has been deemed as a "Write Once, Run Anywhere" kind of language. Since Java is an object-oriented language, it makes it easy to organize and structure code in classes. Java code can be compiled and ran on any platform with a Java Virtual Machine on it without the need to recompile. Once the Java code is compiled into machine code (Java Bytecode), it is then executed on the machine using a JDK (Java Development Kit). Some key concepts of an object-oriented language like Java are classes, objects, encapsulation, inheritance, polymorphism, and abstraction. All of these processes make any project that is developed in Java, reusable, scalable, and easily collaborable. Reusability is important because it allows users to create an object only once throughout the project, and reuse it as many times as needed, without having to rewrite it. Scalability is important because it allows developers to debug easily while testing individual classes separate from one another. This can make a group project easy to scale due to everyone being able to test their own class or object without interfering with other developers' code. Collaboration is important work can be delegated super easily, by giving each developer separate classes and objects, making it easier to work on. Java is mainly used in Web Development, Android mobile applications, machine learning, etc. In 2022, Java is still one of the most popular programming languages, ranking third on the GitHub list. For new developers, Java is super easy to learn and it is secure so developers don't have to worry about compromised data. In conclusion, Java's combination of simplicity, portability, and robust features have ensured its continued relevance in the evolving world of programming languages.

3.1.7.4 Python

Python is a high-level, general-purpose programming language that is known for its simplicity, readability and versatility. It was invented by Guido van Rossum in 1991. It is typically developers first coding language because it is easy to learn and use. Python uses English-like syntax and indentation for code blocks, making it more accessible to beginners. It is also an interpreted language, meaning that Python is executed line by line, making it easier to debug and test. Developing in Python is also super efficient due to the fact that Python doesn't need explicit type declarations. Like Java, Python is also an object-oriented language and supports object-oriented principles as well. Python is also a cross platform language that runs on various operating systems with an extensive standard library that has a comprehensive set of modules and functions that can be used for web development, AI, data science, software development, etc. Since its release, Python has a large and very active community that is constantly growing and

finding new ways to utilize and improve. Overall, Python is a very powerful language that would allow beginners and experienced programmers alike to create strong, robust applications in the fields of data science, web development, and artificial intelligence.

3.1.7.5 GitHub

GitHub is a cloud-based platform that revolutionizes software development collaboration and version control. Founded in 2008 by Tom Preston-Werner, Chris Wanstrath, P.J. Hyett, Scott Chacon and later acquired by Microsoft in 2018, GitHub has become the world's largest source code host, with over 100 million developers. A GitHub repository can either be private, public or open-source, making it super accessible to anyone in the development community. GitHub primarily works using a version control concept. Meaning, that a person can "clone" a repository to make changes to it either on the "main" branch or their own development branch. Once they are done with making changes, they can then upload those changes and "merge" with the existing repository to push those changes and test. This process of merging and creating new versions makes collaboration almost effortless because it allows developers to edit as a group without stepping all over each other's changes.

The GitHub platform uses Git to help maintain versions of repositories. Git was invented in 2005 by Linus Torvalds. It was developed to address the needs of managing the Linux kernel source code, which required a system that could handle large-scale distribution of software. Git helps keep track of changes to files over time by tracking changes as "Commits". Commits are recorded snapshots of the project at a specific point in time. This is extremely useful because if the project somehow becomes corrupt or the code gets messed up somehow, you can actually go back to a previous commit to revert the changes that messed up the project without having to start all over. Some of the benefits of using Git include but are not limited to speed, data integrity, staging capability and flexibility. Git allows users to organize and stage their changes so that developers can review them before sending them off to merge with the rest of the project. Developers can do this efficiently with the commands that Git comes with as well. To ensure data integrity and safety, Git uses SHA-1 hashes to ensure that data doesn't become lost or corrupted. SHA-1 hash is a cryptographic function that creates a 160-bit hash value from a single input. It was designed by the U.S. National Security Agency in 1995 and has since become the standard in data security. Overall, using Github is not only secure, it is also efficient and perfect for all group projects due to its versatility, user-friendliness, and collaborative potential.

3.1.7.6 BitBucket

BitBucket is a cloud-based Git repository system designed by Jesper Nøhr as an independent startup company in 2008 and officially launched in 2010 when it was acquired by Atlassian. BitBucket is very similar to GitHub, where it allows a group of developers to actively work on a software project together. BitBucket also uses version control to help control data inside of repositories. The main difference between GitHub and BitBucket is that BitBucket has other softwares like Trello and Jira built into it that helps developers work more efficiently. Jira helps developers detect bugs easily and provides suggested fixes to the code in order to make the debugging process more efficient. Trello makes cards with tasks for the developer that get added to a calendar that helps tell the group what needs to get done each day. Even though it would make

sense with more features to gravitate towards BitBucket, it does have a cost of use which turns users away from it. GitHub is free to use and is mainly used by the public community to share code and projects with each other while BitBucket, has been adopted by many corporations and private developers to help them stay on top of their tasks on a corporate level. As of 2025, BitBucket continues to evolve, offering features like native security tools, premium platform support, and flexible planning tools.

3.1.7.7 Our Selection: C/C++, Swift/SwiftUI

When it came down to our development environment, the thing that was most important for our group was familiarity. We wanted to keep this project as simple as possible with the resources and technologies that we already knew. The whole purpose of this project was to put together the knowledge that we have already learned throughout our college engineering careers. Essentially, we didn't want to have to learn a whole new programming language or a new repository system when we could have just used one that we all have used before. With this being said, we decided to use GitHub as our main repository system so that way we can all collaborate on this project together. Everyone in our group has used GitHub at least once before, so this will allow an easy transition for this project.

The Github community has an abundance of open source code that will become useful in development for our ESP32. Having GitHub also makes it super easy to transfer files to one another while collaborating, which is what we loved so much about it. As far as planning softwares go, GitHub does not have any of those, but Trello or Jira can always be used separately for helping us plan our project milestones.

For our programming languages, we decided to go with C/C++ and Swift/SwiftUI. For our ESP32, C/C++ will work best and it is also what we are most familiar with, so we can just polish our skills on this project. Everyone has also worked with microcontrollers before using C/C++, so this will make it so much easier to develop a working prototype. For our application, Swift/SwiftUI was an easy option considering that Matthew has worked with them before developing iOS applications. Since SwiftUI is super user friendly with a real-time preview, it'll make developing an application super efficient and easy. SwiftUI also has plugins for allowing bluetooth connections to external devices, which will make integrating the application with our device a breeze. We can also test our application on our own devices since we all have iPhones. This will make the development process more streamlined rather than using an android development software.

3.1.8 Computer Vision

3.1.8.1 OpenCV

Computer Vision is a field of study with a goal that enables machines to gain an understanding of visual information. OpenCV (Open Source Computer Vision Library) is an open-source computer vision library for machine learning that is used worldwide by hobbyists, researchers, and developers. OpenCV's library includes image and video processing, object detection and tracking, facial recognition, and machine learning integration. Computer Vision and libraries such as OpenCV, require loads of data and training to learn and classify images. Classifications made by humans everyday can be

replicated and even surpassed in their accuracy/speed with the help of intensive AI training algorithms in computer vision.

3.1.8.2 Tensorflow and TensorflowLite

Tensorflow is another open-source computer vision library developed by Google that supports a wide range of technologies. Examples of technologies that Tensorflow uses are as follows: neural networks, dataflow graphs, tensor processing units (TPUs), Eager execution, Auto differentiation, distributed computing, and high level APIs. Neural network technologies are applied to tasks like image recognition and natural language processing and are employed by tensorflow's data flow graphs to represent computations. Nodes in these graphs represent mathematical operations, while edges represent multidimensional data arrays (tensors). This flexible architecture allows for efficient execution and processing across GPUs, CPUs, and TPUs. This allows for universal application across various platforms ranging from desktop computers to high-end servers.

A decade ago, researchers at NVIDIA found that GPUs are useful at matrix operations and can be utilized as a supercharger for compute-intensive tasks. This is made possible because of GPUs massive parallel core architecture, allowing for thousands of efficient cores to be launched in parallel threads. As a result, tensorflow runs 50% faster on the latest NVIDIA GPU. In summary, TensorFlow integrates cutting-edge AI technologies such as neural networks, TPUs, distributed computing, and high-level APIs to enable scalable machine learning solutions across various domains. Its flexibility makes it a cornerstone tool for developing state-of-the-art AI applications.

3.1.8.3 Keras

Keras provides tools for creating neural networks using simple, modular components such as layers and models. Keras supports multiple backends, however it is tightly integrated with tensorflow, making it especially powerful for scalable learning machine workflows. Having easy integration with all TensorFlow projects, Keras can be installed easily for implementation with a simple pip command. Acting as a high-level API within TensorFlow, Keras simplifies tasks such as model definition, training, evaluation, and deployment. An API stands for Application Programming Interface and functions as a set of rules or specifications that allows different software applications to communicate with one another, sharing functionality. Keras supports various data formats such as NumPy arrays, and Pandas DataFrames, integrating seamlessly with TensorFlow datasets. In summary, Keras serves as a bridge between high-level model development and efficient execution through TensorFlow. Its integration into platforms like Teachable Machine demonstrates its versatility for both beginners and advanced users in machine learning workflows.

3.1.8.4 YOLO

YOLO stands for "You Only Look Once" and is a real-time object detection algorithm that processes an entire image or frame with a single pass through a neural network. This means that YOLO treats object detection as a regression problem, predicting both the bounding boxes and classes of objects simultaneously. The algorithm works as such, an image is taken or received, divided into a grid of cells, each cell's bounding box coordinates and prediction is made, overlapping predictions are filtered to maintain most

confident predictions, and finally the objects are returned with their most confident bounding boxes and confidence scores. This single-stage object detection method makes YOLO an algorithm applicable to real-time processing speed scenarios. Strengths of YOLO include processing images at around 150 frames per second, single-pass architecture proves higher efficiency than other algorithms, and integrates with frameworks like PyTorch and TensorFlow. On the other hand, YOLO's single-pass architecture shaped for efficiency and speed has its weaknesses. YOLO's grid based approach can struggle to detect small or multiple objects, sacrificing accuracy for speed. However, YOLO remains one of the most widely used object detection algorithms due to its unmatched speed and versatility in real-world applications where rapid decision-making is critical.

3.1.8.5 Scikit-image

Scikit-Image is an open-source Python library for image processing and computer vision. It is part of the broader scientific Python ecosystem and is built on top of foundational libraries like NumPy and SciPy. Scikit-Image provides a wide range of tools for manipulating, analyzing, and enhancing digital images, making it a popular choice for researchers, engineers, and developers working on image-related projects. The library emphasizes accessibility, offering a user-friendly API that integrates seamlessly with other scientific Python tools like Matplotlib and Scikit-Learn. It supports tasks such as filtering, segmentation, feature extraction, morphological operations, and geometric transformations. Scikit-Image is widely used in fields like medical imaging, astronomy, robotics, and more.

Framework	Capabilities	Ease of Use	Speed	Accuracy	Best Use Cases
OpenCV	Comprehensive library for image/video processing, object detection, tracking, facial recognition, and machine learning integration	High (simple APIs)	Fast	High	Real-time applications involving color/shape labeling
TensorFlow/ TensorFlow Lite	Advanced deep learning framework supporting neural networks, TPUs, dataflow graphs, and distributed computing. TensorFlow Lite enables deployment on mobile/edge devices	Moderate	Moderate	Very High	Deep-learning based object detection, facial sentiment analysis
Keras	High-level API for building neural	Moderate	Moderate	Very High	Custom object detection tasks

	networks with TensorFlow backend, simplifies model creation, training, and deployment. Supports integration with datasets like NumPy arrays and Pandas DataFrames.				that require high precision
YOLO	Real-time object detection algorithm that processes images in a single pass through a neural network. Optimized for speed, but struggles with small/multiple objects due to grid-based architecture.	Moderate	Very Fast	High	Real-time application in surveillance, human posture detection, and self-driving cars
Scikit-Image	Filtering, segmentation, feature extraction, morphological operations, and geometric transformations.	High	Moderate	Moderate	Simple image processing tasks including basic object/face detection

4. Standards and Design Constraints

4.1 Industrial Standards

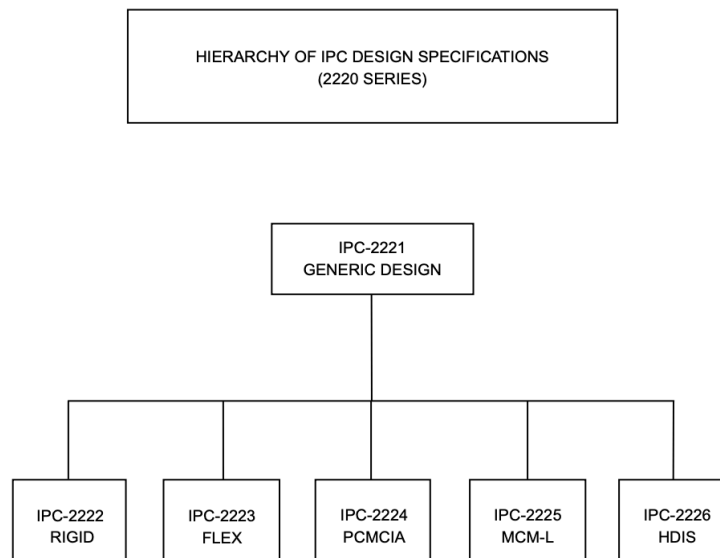
4.1.1 PCB Design Standards

Printed circuit boards (PCBs) are everything in the electronics world. They support everything from TVs to wearable smart watches. While PCBs are essential for everyday life, how can we ensure that our devices and machines work as expected? Standards. The Institute for Printed Circuits (IPC) was developed to support electronic consistency and safety in manufacturing PCBs. They have developed a set of standards that have become known as the IPC standards that ensure manufacturers create safe and consistent products. The IPC has over 300 standards that cover the entire PCB lifecycle, not just the manufacturing processes.

For this project, our PCB manufacturer of choice is going to be JLC PCB, so their standards are going to be used. Their standards include maximum and minimum thickness, clearness, required diameters and required export file formats. Our manufacturer has a wide range of capabilities for this project's Rigid PCB design. JLC PCB allows anywhere between 1-32 layers of copper with a choice of four materials, FR-4, Aluminum-Core, Copper-Core and RF PCB. The dimensions can range from 3 mm x 3 mm - 670 mm x 600 mm. Essentially, you can make it as big, or as small as you want since the dimensions that they allow are such a huge range. Depending on the material and layer count, you can range from 0.8 mm - 4.5 mm of thickness in your PCB. They offer a variety of colors for the solder mask such as green, purple, red, yellow, blue, white, and black as well to suit any project. When it comes down to the drilling of the PCB, JLC PCB can range from 0.3 - 0.63 mm for one layer, 0.15 - 0.63 mm for two layers, and 0.15 - 0.63 mm for 2+ layers. The via hole-to-hole spacing is 0.2 mm with a pad spacing of 0.45 mm. For more experienced engineers, you can make your traces as small as 0.1 mm for one layer, and 0.16 mm for 2 layers. The minimum distance from track to the edge of the board is 0.3 mm.

IPC-2221 guidelines are for the design of PCBs. This standard involves a wide range of aspects related to PCB design to ensure reliability, feasibility, and performance of all PCBs.

Figure 4.1.1.1 Hierarchy of IPC Design Specifications



IPC-2221 Generic PCB Design Specifications addresses many different guidelines to ensure that quality, safe, and reliable PCBs are designed and manufactured. Some of the key points that IPC-2221 addresses are:

- **Design Requirements:** IPC-2221 establishes fundamental design principles for various types of PCBs, including rigid, flex, and hybrid boards. These principles and guidelines are essential for achieving proper layer stack-up, trace routing, component placement, and power management.
- **Material Considerations:** The IPC-2221 provides guidance on selecting PCB base materials, including laminates, substrates, and surface finishes. It considers properties like dielectric constant, thermal conductivity, and moisture absorption to provide the best recommendations for material selection to meet the performance requirements for the electronics under different conditions.
- **Electrical Performance:** IPC-2221 covers specifications such as signal integrity, impedance control, and electromagnetic compatibility (EMC). It also defines standards for trace width, spacing, and controlled impedance. It ensures that there are no stubs and sharp angles in traces to ensure signal integrity. To achieve high-speed PCBs, IPC-2221 uses differential pair routing to transfer high-speed signals via different platforms like USB, HDMI, and Ethernet in order to achieve a clear return path for signals over continuous ground planes. This is a critical area in the IPC-2221 standards to ensure reliable, high-speed PCB functionality.
- **Mechanical Considerations:** IPC-2221 defines standards for PCB manufacturing. It creates standards for PCB board thickness, hole size, and aspect ratios. Standard board thickness has been defined as 1.57 mm, but you can use other thicknesses based on number of layers, copper weight and the expected stress of the application on the PCB. The IPC-2221 defines minimum and maximum diameters for holes in order to ensure reliable plating and structural integrity. It also covers mechanical constraints such as warpage, bow, and flexural strength. It sets limits on the acceptable warpage to ensure the PCB remains as reliable and as optimal without any warpage. Under mechanical considerations, it also covers topics for cutouts and slots, PCB board edge clearances, and guides for mounting holes. This is essential for determining that our PCB will be physically durable and manufacturable.
- **Thermal Management:** Heat and temperature of PCBs are essential to make sure boards don't overheat and that our components dissipate that heat as best as possible. IPC-2221 offers recommendations for heat dissipation, thermal vias, and component placement based on different conditions. It also addresses design considerations for thermal expansion and contraction which will help not cause damage to components when they are under stress. Thermal management is essential to ensure reliability in everyday devices.
- **Manufacturing and Fabrication Guidelines:** IPC-2221 covers design rules for manufacturability (DFM) to ensure cost-effective production. It talks about how easy the fabrication, assembly and testing can be when the rules are followed. It defines hole sizes, pad geometries, and drill tolerances for various PCB classes

to reduce the likelihood of manufacturing problems and ensures efficiency of the overall production process. It has become one of the most critical concepts for PCB design to this day. The DFM in the IPC-2221 gives engineers tips and guides them in order to ensure their fabrication process goes as smoothly as possible, without any errors or setbacks.

- **Plating and Finishing:** The IPC-2221 details requirements for plating thickness depending on the material (copper, tin, gold, etc.). It also has rules in order to prevent oxidation of certain materials and improve the solderability of your PCB to increase its overall lifespan. Based on the material, it covers surface finishes such as HASL, ENIG, and immersion silver to ensure electrical performance and enabling long term corrosion resistance.
- **Reliability and Testing:** The IPC-2221 gives engineers recommendations on design practices that support long-term reliability. It establishes criteria for electrical testing, thermal cycling, and mechanical stress tests. It outlines environmental considerations for optimal testing such as humidity, temperature extremes, and vibration. This will help make sure the PCB will endure its intended operating environment. This section talks about different reliability classes ranging from 1-3. 1 being general electronics, 2 being electronics for industry and 3 being high-reliability electronics for the medical, aerospace and the military. Each reliability class has different standards that the electronics need to meet in order to remain reliable to its intended use. When it comes to testing, there are different categories of testing, ranging from electrical testing, to chemical and material reliability. The IPC-2221 have standards to run tests until complete technological failure. This is to ensure that these products are ready for the everyday environment and to take information to improve the next generation of products.
- **Component and Soldering Considerations:** The IPC-2221 provides engineers with guidelines for land patterns, annular rings, and pad design for through-hole and surface-mount components for PCBs. It helps ensure compatibility with soldering processes like wave soldering and reflow soldering.
- **Design for Assembly (DFA) and Design for Test (DFT):** The IPC-2221 recommends best practices for component orientation, spacing, and accessibility. This helps manufacturers with the fabrication process to ensure all components fit properly and there are no issues with installing the components. It also specifies test point placements for in-circuit testing (ICT) and boundary scanning to ensure PCBs are designed for real-world application testing and not for just simulation. This will assist in faster assembly, lower production cost, troubleshooting, etc.

Overall, the IPC-2221 serves as a complete guide for PCB design, providing a key framework for creating reliable, efficient and durable electronic systems. These standards will be utilized in this project as a guide for us to send a PDB design to be fabricated at minimal cost and receive a properly functioning, and reliable PCB.

4.1.2 UART Communication Method Standards

The Universal Asynchronous Receiver/Transmitter (UART) is an asynchronous serial communication interface that is found in nearly every embedded system, MCU, and peripheral device. Unlike other complex protocols such as USB or Ethernet, UART does not define a rigid protocol stack. Instead, it defines a method for sending and receiving serial data, with the electrical characteristics and standards of the industry. Over the years, several standards have become well established within the industry. Standards such as RS-232, RS-422, and RS-485, that define both the electrical and mechanical properties of UART-based systems.

For starters, UART communication is based on a simple data structure that includes one start bit, five to nine data bits, an optional bit for error detection, and one to two stop bits (Refer to *Figure 3.1.6.1.2*). This allows for asynchronous communication between two devices without requiring a single, shared clock signal. For UART to work, both the transmitting and receiving devices must be configured with identical communication parameters (baud rate, data length, parity and stop bits) for successful transmission. In UART, the start bit signals the beginning of a data frame, followed by the payload, then the error detection bit, then the stop bits to indicate the end of the data transmission. This framework structure allows UART to be extremely reliable in a multitude of systems ranging from slow, low-speed debugging consoles to fast, high-speed GPS and Bluetooth devices.

While UART defines the actual communication format, the actual physical and electrical signaling is handled by other external standards. The most common among these standards is the RS-232, which deals with voltage levels and different connector types for serial communication. UART signals need to be converted to RS-232 voltage levels using line drivers and receivers. RS-232 transmits a logic '1' as a voltage indicator anywhere between -3V and -15V and a logic '0' for voltages anywhere between +3V and +15V. This goes to show that RS-232 effectively uses negative logic compared to typical digital logic levels. RS-232 is specifically designed for point-to-point connections for distances up to 15 meters with baud rates up to 115.2 Kbps or higher. RS-232 has been widely adopted for PC serial ports and remains useful for major industrial and government equipment.

Sometimes, data needs to be sent over long distances. To solve this, UART often uses RS-422 or RS-485 standards. The RS-422 standard only supports one transmitter with multiple receivers, while the RS-485 standard supports multiple transmitters with multiple receivers on the same differential pair of lines, allowing for multipoint communication. These standards use differential signaling, which is beneficial for reducing noise and increasing distance. These signals support higher data rates than single ended standards like RS-232. RS-485 is often found in automation, control systems, and other environments where long distance communication is crucial.

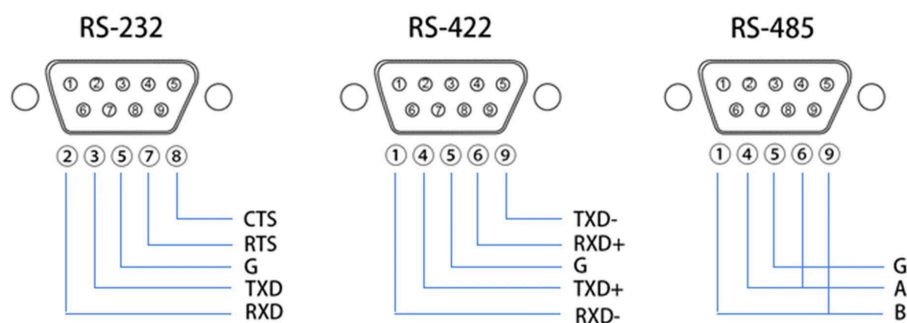


Figure 4.1.2.1 Comparison of UART Standards

In today's modern embedded systems, UART is used in the logic-level form or TTL UART, where communication occurs directly between integrated circuits that are using standard logic voltage levels such as 5V, 3.3V or 1.8V. This type of UART, often used between a MCU and its peripherals, does not conform to the RS-232 voltage levels and can't be directly connected to RS-232 ports without performing some kind of level shifting. TTL UART is simple, efficient, and effective, but careful action must be taken to match the voltage levels between devices in order not to damage them.

Since UART has no control mechanism built in, systems will often implement flow control in order to prevent data loss during high-speed or buffered communication scenarios. For hardware, control flow used additional lines such as RTS (Request to Send) and CTS (Clear to Send) in order to manage data transmission. These lines are typically a part of the RS-232 specification. On the other hand, software flow control uses control characters such as XON/XOFF, which are embedded into the data stream to pause and resume data transmission. Flow control is especially important in systems where one device has a limited buffer capacity or is extremely slower than the other.

There is not a standard baud rate that is defined by UART, but among the community, there has become a set of commonly accepted values for different hardware and operating systems. These include 1200, 2400, 4800, 9600, 19200, 38400, 57600, and 115200 bps. There are even higher baud rates like 230400, 460800, and 921600 which are available on more modern devices. Most peripherals for UART allow for flexible configuration through internal clock division or dedicated baud rate generators.

While UART does not define protocols beyond its framing, higher-level protocols such as AT commands and Modbus RTU are often layered on top of UART links in order to enable structured communication. For more demanding and modern applications, there are additional standards such as IEEE 1149.1 (JTAG) for boundary scan testing or for SoCs (System on Chips) and other modern technology.

In conclusion, while UART isn't governed by a single protocol or standard, its operation is shaped by widely accepted conventions and physical layer standards such as RS-232, RS-422, and RS-485. These standards help define voltage levels, cable requirements and signal types. This will ensure that UART communication is as robust and versatile as possible. By understanding these widely accepted conventions, along with the best practices for testing, framing, flow control, and signal integrity, designers can effectively implement UART in both simple and complex systems.

4.1.3 I2C Protocol Standards

As mentioned in Chapter 3, the I2C protocol, is a widely adopted, synchronous, multi-master, multi-slave, serial communication protocol developed by Philips Semiconductor in the 1980's. It was designed for integrated circuits to communicate on the same PCB, making it essential for slow, short distance communication within the same embedded system.

I2C operates over two bidirectional open lines: the Serial Data Line (SDA) and the Serial Clock Line (SCL). These lines are connected to a common supply voltage (typically 3.3 V or 5 V) and are pulled up. The devices on the bus can pull the line down, but they can't

drive it high. This will ensure that the communication between multiple devices is safe and there are no serial conflicts. This protocol supports multiple masters and multiple slaves, meaning that any master device can communicate and transfer data to any slave within the system.

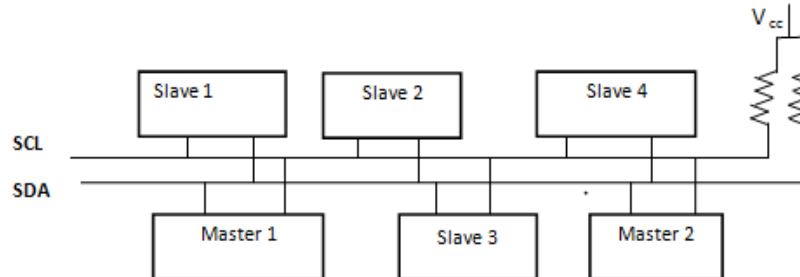


Figure 4.1.3.1 I2C Multi-Master Multi-Slave Communication

Since I2C communication is synchronous, each data transfer begins with a start condition (high-to-low transition on SDA while SCL is high) and ends with a stop condition (a low-to-high transition on SDA while SCL is high). Data is transferred in packets of 8 bits (1 byte) followed by a 1 bit acknowledgment (ACK/NACK). A 7 bit address field has become the most common format because it allows up to 127 different slave devices on a single bus. For larger systems, an extended 10 bit addressing mode is supported (Refer to Figure 3.1.6.3.2).

The I2C protocol has developed a standard for several bus speeds:

- **Standard Mode:** Up to 100 Kbps
- **Fast Mode:** Up to 400 Kbps
- **Fast Mode Plus (FM+):** Up to 1 Mbps
- **High-Speed Mode (HS-Mode):** Up to 3.4 Mbps

What's great about I2C devices is that the devices that support different speed modes can coexist on the same bus. This is of course if the master initiates communication at the appropriate speeds depending on the device. However, all devices must support the lowest common speed if switching mechanisms are not used.

Electrical characteristics and timing requirements for I2C are defined and maintained by NXP Semiconductors in the I2C-bus specification. This specification outlines detailed constraints on rise/fall times, bus capacitance, pull up resistor values, and certain voltage thresholds. For example, the total bus capacitance should never exceed 400 pF because it limits the cable length and number of devices that can be added. Choosing pull-up resistor values (typically between 1 K Ω and 10 K Ω) is crucial for maintaining reliability and signal integrity.

In the end, the I2C protocol is set by a well-defined standard that allows for reliable, low-cost, two-wire communication among chips on the same board. I2Cs use of multiple addressable slave and master devices, along with its simple hardware interface makes it highly versatile. Whether in small networks or in complex systems, I2C remains a fundamental tool for embedded systems development.

4.2 Design Constraints

4.2.1 Time

For this project, one of the main constraints is time. Since this project is a Senior Design only project, it gives us only a certain amount of time to complete it by fabricating our idea and design to bring our vision to life. Due to this, there are some decisions that must be made with time in mind that will help us meet deadlines rather than taking longer to build a better project but possibly not completing the project.

One crucial issue was the time constraint because we were limited to one academic calendar year to make a fully functional, flawless prototype. Meaning, there are no extensions on the timeframe, giving us hard deadlines that we need to meet in order to pass the class and move on to graduation. Lucky for us, we already had our group planned out from a semester in advance and we had a general idea about what we wanted to do. We started doing research and making sure our idea was feasible in this time frame beforehand in order to get ahead of everyone else.

Since software is involved in our project, we will run into another issue with time due to all of the debugging and testing that will need to be done. When it comes to software and programming, you never know how long a problem is going to take to get solved. This will cause multiple setbacks that must be fixed within this short time frame that we are given. Meaning, these unforeseen errors will pose a massive threat to our deadline and we will need to work efficiently to take care of these errors as fast as possible.

An added issue that may affect this project is ordering and shipping of parts. Especially from JLC PCB, where the shipping can take anywhere from 4-6 weeks. We will need to test and debug using breadboards in the mean-time so that we know our software works and when the PCB comes in, we can then put all of the components on it and just simply upload the working code. If we don't test and debug during the shipping of the PCB, it can lead to massive problems with time management, especially with a hard deadline.

3D printing has become common among Senior Design projects. Since we are planning on 3D printing the housing for our device, we need to take into account the fabrication time and account for printing errors, especially since no one in our group has prior 3D printing experience. This can lead to us wasting valuable time printing and designing when we can be debugging or soldering components on the PCB.

The result of all of these time constraints can lead to rushing a project. Rushing can cause us to cut corners and not give the expected results that we want. Time management, accountability and communication need to be on point for all group members so that way we can meet the goals that we have outlined for each other and this project as a whole. If you say "I will do it" then you better do it by the expected deadline so that way the group doesn't fall behind. We need to hold each other accountable for our work. By not doing these things, it can lead us down the path of failure by not meeting the hard deadlines that have been set for us.

4.2.2 Economic

For SafeSight, one of the most challenging constraints was the economic constraint. Since we are all college kids, and don't have the funds to build the most expensive, best performing products, it made it difficult because we had to decide if we wanted a cost-effective product or a better performing product. Specifically, for the radar sensor, there were many different options that involved varying ranges for detection. There was a hard tradeoff between range and cost in order to make sure that we don't break the bank.

Another economic constraint was that we needed to take into account if any of our parts break, we need to buy a new component. We may not take that into account with our budget, so we need to allot time and money in case we run into that roadblock. By doing this, it allows us to not go over both time and money.

When researching parts, we came across another economic constraint, the Raspberry Pi. We ran through many different subsystems for the computer vision aspect of this project. Our first idea was to use a NVIDIA Jetson Nano Super which runs about \$249.00. This was obviously way out of our budget range and we decided to go with something a little more budget friendly, the Raspberry Pi V5. The difference between the two is that the Jetson Nano excels in AI and computer vision tasks due to its NVIDIA GPU, while the Raspberry Pi V5 is more for general use with a focus on speed and versatility. The Raspberry Pi V5 runs about 90 dollars and depending on the peripherals that we need, it may end up costing us around \$130-\$150, which is much more affordable than the Jetson Nano, and could still do everything that we need it to do to handle our computer vision for this project. With the Pi's versatility, it will allow us to add many different peripherals without having to worry about if the device is compatible or not. This will cause us to save time and money in the long run.

Just like the Raspberry Pi V5, another piece from the parts selection that we needed to consider was the microcontroller unit of our project. The first option that came to mind when making this selection was to use an Arduino microcontroller that has built-in Wi-Fi to connect to the phone for our application use. However, there were very limited options when it came to those requirements that would fit our budget. We decided to find a microcontroller with both Wi-Fi and Bluetooth interface so that way we SafeSight can communicate with our application with ease to view pictures and files from Bluetooth rather than Wi-Fi. We stumbled across the ESP-32 and realized that it would be perfect for our project since it had Wi-Fi and Bluetooth with a fast enough CPU to get the job done. Buying an Arduino MCU, specifically the ATmega328p, would've just been a waste of money since it did not have the specifications that we were looking for. The ESP-32 costs about \$12 for a three pack while the ATmega328p costs around \$9 for one. This benefits us because in case we break an ESP-32, we don't have to waste both time and money buying a new MCU. Therefore, because of this economic constraint, we decided to pick the ESP-32 since it more accurately matched our needs.

In conclusion, the economic constraint poses a massive threat to our project because it causes us to pick products that are more budget friendly rather than favoring performance. This means that there needs to be a lot of research that needs to be done so that we don't run into challenges along the way. We could also help with the economic problem by throwing in a little extra money along the way to help increase our budget to help overcome any financial problems that we may encounter.

4.2.3 Federal Regulation Constraints

Developing driver safety technology to be mounted on a windshield involved adhering to several federal regulations and standards to ensure safety, functionality, and compliance. The Federal Motor Vehicle Safety Standards (FMVSS) dictates some requirements in regards to the windshield mounting aspect of this system. Firstly, there are retention standards that include the windshield remains retained on the car in any aspect of a collision or sudden change of force applied to the windshield. This directly correlates to the device as it must not intrude on 50% of the windshield periphery, given the vehicle has passive restraints on the windshield and 75% periphery for vehicles without passive restraints. The device must also not obstruct the area swept by windshield wipers, as this can compromise visibility during adverse weather conditions. Under section 108(a)(2)(A) of the FMVSS regulation text it states : *“manufacturers, distributors, or repair businesses are prohibited from rendering any safety feature inoperative.”* Meaning that our device cannot interfere with windshield functionality or any other safety systems like airbags or lane assistance in more up to date vehicles.

Federal Motor Carrier Safety Administration (FMCSA) regulations also add onto the federal regulations expected in this system. One regulation is the placement of the device in the vehicle. FMCSA regulations define specific areas where devices can be mounted on windshields, these areas are restricted to no more than 8.5 inches below the upper edge of the windshield wiping area and no more than 7 inches above the lower edge of the area swept by windshield wipers. The bright side of the FMCSA regulations is that they do allow mounting devices on the windshield as long as they are explicitly designated “vehicle safety technology” which includes systems like lane departure warnings, braking assist systems, driver cameras, GPS units, traffic sign recognition, lidar, radars, and sensors. The system being designed meets these specifications.

National Highway Traffic Safety Administration (NHTSA) guidance tells requirements for the device being developed to acquire correct certification. The manufacturers, in this case being the team developing the product, must comply with all applicable FMVSS before being sold or installed in vehicles. This includes the devices not hindering existing safety standards like crash resistance or visibility. Given that the device will be installed after the vehicle is purchased from a car dealership, the installation process must not compromise compliance with FMVSS standards for new vehicles or equipment. Some examples include not interfering with windshield integrity or other factory installed safety features. As windshields also have their own regulations such as coloring, obstruction prohibitions, and decals or stickers, mounted devices must not violate these specifications as well.

Additionally, there must be testing conducted to ensure that the method of keeping the device in place is effective in a collision scenario. Given a collision, the device cannot result in a projectile, must withstand dynamic shock and vibrations as well. The device must minimize distractions by ensuring the drivers do not look away from the road for more than 2 seconds as this will undoubtedly cause the system to function inversely from its purpose.

4.2.4 Remaining Constraints

4.2.4.1 Health and Safety

Another constraint that must be factored in when developing and testing this project is health and safety. Since this project is going to go inside people's cars, we need to make sure that the product is reliable. The safety of people while driving is our top concern and the last thing we want is a device that doesn't work properly or is slow. For example, we don't want the device to pick up a green light by mistake, alert the driver, the driver go into the intersection, and then crash because it wasn't a green light. As a group, we need to make sure our device is reliable and accurate so that way it isn't giving false results to the driver.

We had to also take into account how we were going to be powering the device. The last thing we wanted is to power it via batteries, and under intense heat, have the batteries explode and cause a fire in the car. The safety of the drivers was taken into account after we realized that most people keep their cars outside. As we all know, keeping a battery in direct sunlight is no good. That is why we decided to power it via the 12V cigarette lighter in the car. That way, there is no potential for a battery related fire, keeping the drivers safe at all times.

4.2.4.2 Manufacturability

The final constraint that is probably the most important for this project is manufacturability. Our group is limited to the lab equipment that is provided to us for Senior Design students only. This limits the amount and type of manufacturing that we can do severely. This means that we need to keep our design simple enough to be fabricated in the lab.

Since we are allowed to buy components online, that means that we are not fabricating any MCUs by hand. This means that we can just simply put the components that were purchased online together and make them communicate. The only thing that needs to be truly fabricated is the PCB and that gets sent out to JLC PCB to get done over there. However, once we get it back, we need to solder all the components onto the PCB, which will factor into the manufacturability and feasibility constraint for this project.

5. Artificial Intelligence Comparison

5.1 ChatGPT

ChatGPT has become known as one of the worlds best tools in regards to conversing, creativity, and problem solving. This human-like conversationalist is amazing as a sidekick where we as humans can ask it problems pertaining to detailed examples we come across and it can work with us to solve issues like debugging code, explaining concepts, making images, and much more. Throughout the progress of the project, ChatGPT was used to refer to spec sheets that gave us hardware pinouts, example code in order to flash the hardware correctly, and ideas on how to approach certain wiring and applications to make the system work as intended. For example, in regards to conducting research on this system we are designing, chatGPT offered valuable insight on what to research in regards to limitations, compatibility, and functionality of all components. Alongside the brainstorming capabilities of ChatGPT, there were also uses for this LLM in regards to integrating hardware at the testing bench. ChatGPT offered valuable insight on how to program the ESP32 through the Arduino IDE for example, which gave us the rundown of required files, example code, and synchronizing processing data into the ESP32 board.

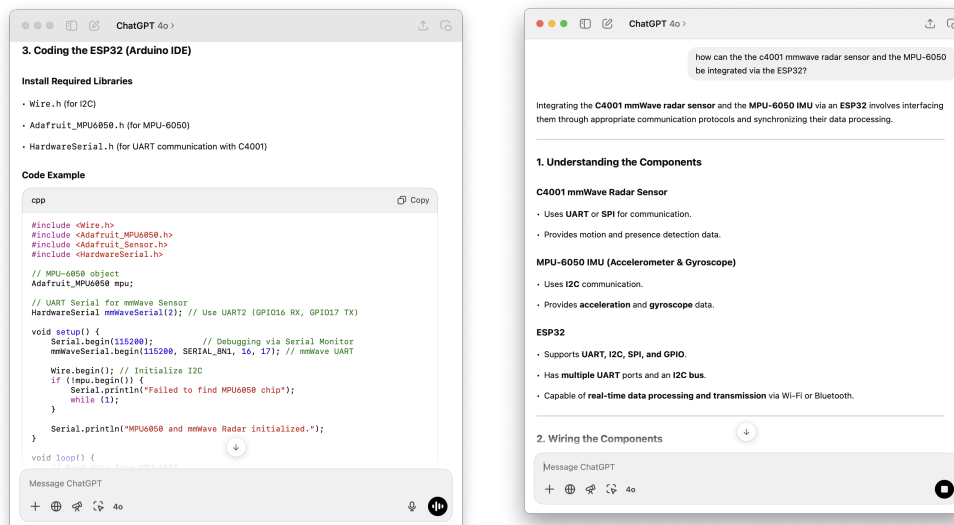


Figure 5.1.1 ChatGPT usage example

From the ideas that ChatGPT gave insight on, another LLM used was Perplexity. Perplexity is an Ai powered search engine that gives us real time information that then provides sources. These sources were often educational texts, hardware manufacturing firms, and engineering specific websites that often encountered experiences with these components.

5.2 Perplexity

For the majority of research conducted on compatibility, meeting constraints, identifying what components were going to function according to the specifications of the system, etc. Perplexity offered a very good platform to identify all of these requirements with research backed to prove its answer. Much like searching on the web through engines like Google, Perplexity was able to search and find direct, reputable sources for the information it provided in accordance with the prompt. This engine also offers features to expand knowledge in which it provided follow up questions to provoke thought and thus led to more questions being answered simply from the Ai's recommendation.

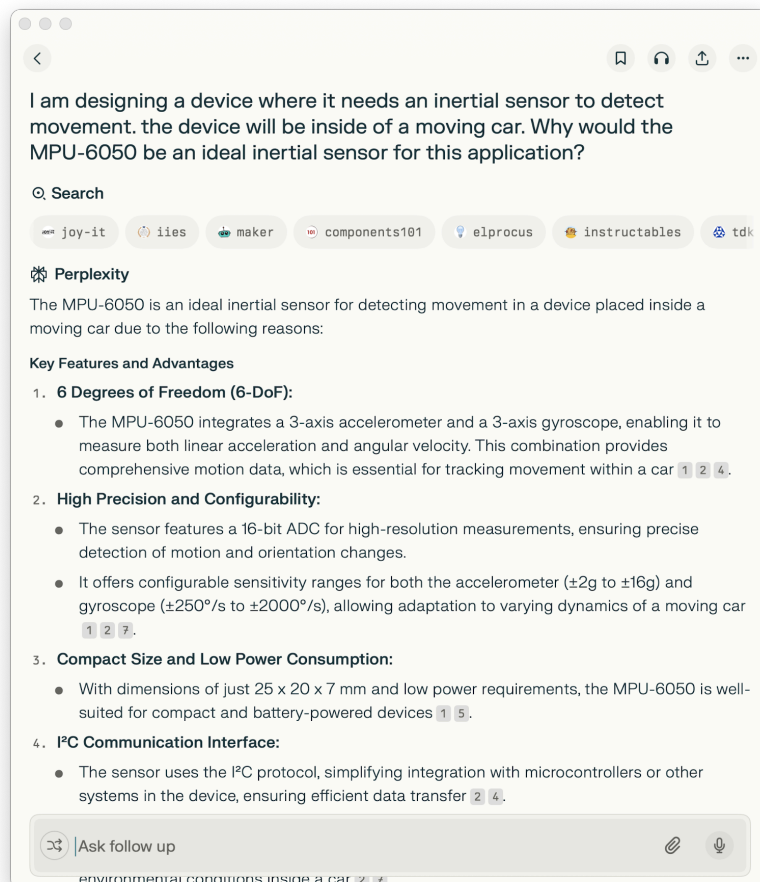


Figure 5.1.2 Perplexity Ai usage example

Here we asked the question on a hardware piece we already decided on. This engine then gave a cohesive list of functional advantages for the inertial sensor. Bringing real information from sources cited at the end of the information provided in the numbered grey boxes.

☰ Related

- Q How does the MPU-6050 compare to other inertial sensors in terms of accuracy
- Q What are the main advantages of using a 6-DoF sensor like the MPU-6050 in a car environment
- Q Can the MPU-6050 handle high-frequency data without losing accuracy
- Q How does the MPU-6050's power consumption affect its usability in battery-powered cars
- Q What are the typical applications of the MPU-6050 in automotive technology

These questions are what the Ai would “guess” you want to know more on, provided added information on topics we may not have thought to cover in the expenditure of our research. This LLM was very useful and proved to be a helpful research assistant where the user, us, was able to pick and choose what information was valuable, similar to the process of using a search engine the traditional way. Except, this LLM allowed for quicker navigation, source identification and summaries of the information you can expect to find in the sources.

6. Hardware Design

In regards to the whole system there is a list of physical constraints we need to fulfill in order to make sure the system is operable within the environment of a moving vehicle. First, we must make the system fit behind the rear view mirror of a car without obstructing the view of the driver. In order for this to be done we went ahead and referred to the guidelines by the Federal Motor Carrier Safety Administration (FMCSA) which dictated that a device in vertical position must not exceed up to 8.5 inches below the upper edge and up to 7 inches above the lower edge of the area swept by the windshield wipers. The device must also remain outside of the driver's sight lines to the road, highway signs, and signals. The device cannot be placed in areas swept by the windshield wipers unless they are explicitly defined as "Vehicle Safety Technology". Fortunately for the system, the device is defined as such.

The FMCSA has broadened its definition of "vehicle safety technology" to include systems such as Camera, Lidars, Radars, Sensors, and video systems for collision mitigation, lane departure warnings, GPS, and traffic sign recognition. The reason these regulations exist is to balance technological advancements in vehicle safety systems with driver visibility and road safety. Mounting devices outside of these prescribed areas or obstructing sight lines can lead to violations and increased risk of distracted driving, which in turn would counteract the purpose of this device we are developing.

Secondly, the device must be durable and robust enough to handle environmental resistance. In the environment of traffic, there are multiple factors that can affect the device. When you turn the car on, its engine begins to provide vibrations throughout the whole vehicle (unless electric). When the car comes to a sudden stop, the rapid change in velocity makes the device prone to heavy change where the force moving forward will overcome the force moving against it. Given the change in seasons we experience on earth, the device must also withstand heavy temperature readings from the sun during warm months and also colder temperatures sometimes below 0 degrees Fahrenheit. In order for these conditions to not affect the system being developed, it is imperative for the components in the hardware design to all have ideal temperature ranges, shock withstanding, and proper solidified connections so that the system does not come apart in the rapidly changing environment that automobile traffic can provide. These considerations bring about another aspect of the mounting mechanism needed for such a system to properly withhold the environment. The mounting system should be secure yet easy to install and remove without damaging the windshield. It should resist the said vibrations and movement while maintaining proper alignment with the design of the system.

In regards to the hardware itself, designing the PCB will also require these considerations. When designing a PCB for use in an automobile, it is crucial to select materials capable of withstanding the shocks, vibrations and harsh environmental conditions to this dynamic environment. Commonly used PCB materials include the FR-4. Since the FR-4 has both standard and high-tg variants, it gives us a range which will be constricted by budget in order to make this system rest on a robust board. These boards are printed on a fiberglass-reinforced epoxy laminate material which is widely used in PCBs. This reinforcement provides high mechanical strength and rigidity, high temperature variants (high tg) can withstand temperature ranges up to 150 deg. C (302 deg. F), and it is cost effective for the budget. The one limitation for this is that the vibration resistance is moderate, in order to work around this we would need to encapsulate the board in a case, as we planned to do so already. The hardware being implemented

on this board will also meet all the shock and temperature robustness described. The recordings of those specifications are provided in the sections pertaining to those IC's as it is a reason for the decision process on the components.

There are multiple mounting systems that can be implemented in order to withstand these environmental conditions. First and foremost, the most ideal would be a velcro mounting system. This system offers heavy duty velcro straps that are used to attach devices securely to mounting surfaces, in this case the windshield. These advantages are that it is flexible and adjustable, allowing for repositioning and removal without the need of extra tools. This system has proven resilient against vibrations and shocks in dynamic environments and also it is extremely strong and durable, capable of holding significant weight (2 square inches of velcro can hold up to 175 pounds). The rearview mirror replacement mounts that attach to the windshield are using the same mounting button as a rearview mirror. This system utilizes an existing mounting point, avoiding adhesives and drilling, while providing a stable attachment point for devices. Since the rearview mounting system is commonly used for backup cameras, monitors, and other devices, it would be ideal for the system to be mounted as such, combined with the velcro in order to ensure shock prevention and even handle scenarios where the device remains secure in a crash or accident.

Third, it is imperative for the design of this system to be appropriate for human-machine interaction (HMI). The interface should be intuitive and easy to understand all while minimizing driver distraction. This includes clear visual and audio indicators for warnings on collisions and distractions. Avoiding the use of extra peripheral devices for use, avoiding complex menus and prolongs attention interactions. As this system will be communicating to the driver on various conditions to be alert on, it is also important for the information to be conveyed in a concise and relevant manner to avoid cognitive overload. This would then bring us to follow the National Highway Traffic Safety Administration's (NHTSA) guideline for in-vehicle electronic devices. These include but are not limited to avoiding the driver looking away from the road for more than 2 seconds at a time, and minimize manual inputs during operation. As this system will function without the need for a driver to input anything after installation, this requirement is met. As Well as the system's implementation to be a simple audio signal from a piezoelectric buzzer, the system will ensure the driver's attention will remain on the road with no distractions being met. Alongside these design features, the method of powering and communicating data with the driver will also be minimal to reduce the space of hardware in the car, removing the concern of the system interfering with the driver's use of the vehicle. Since the car's 12V cigarette lighter port will be used to power the device, it is simply a matter of cable management that can be routed to avoid any interference with the shifting gear and the buttons accessible to the driver. The ESP32's functionality of bluetooth will transmit data from the device to the user's application seamlessly without additional hardware as well. This will be ideal in keeping the system from interfering with the driver's focus on the road.

6.1 Power Supply Design - Regulators

The device with its power demands exceed the probability of having a wireless approach that involves operating on a rechargeable battery. The Raspberry Pi 5 running computer vision tasks in parallel with the ESP 32 consumes an average of 7.2 Watts an hour,

making it so that the battery small enough for a space conscious design would be depleted every week (5-7 hours of use). Thus, a wired design for power that taps into the battery of a vehicle using the cigarette lighter port is the most power conscious and efficient design to power the Safe Sight device.

In addition, keeping our source of main power in mind (12 V car battery), we must also consider the operational power requirements of our ESP 32 (3.3V) and Raspberry Pi (5V). These constraints require us to design regulators that convert and drop down the main 12V source to 5V and 3.3V to properly power the individual operational processing components and sensors in our device. First, we must design the first regulator to step down the 12V voltage of a standard cigarette lighter port in a vehicle to 5V and then make a decision to either have the second regulator tap from the 5V power and convert it to 3.3V or straight from the 12V source (12V to 3.3V regulator).

6.1.1 ESP 32 Power Supply Regulator - TPS564252

The TPS56425x is a line of regulators that take 3V-17V inputs and can convert them to a range of low voltage outputs 0.6V - 10V at a 2-A continuous current. The TPS564252 supplies a steady 3.3V output with a great operational efficiency at a 12V input, making this model in the TPS line of regulators meet the specifications of our ESP 32 perfectly.

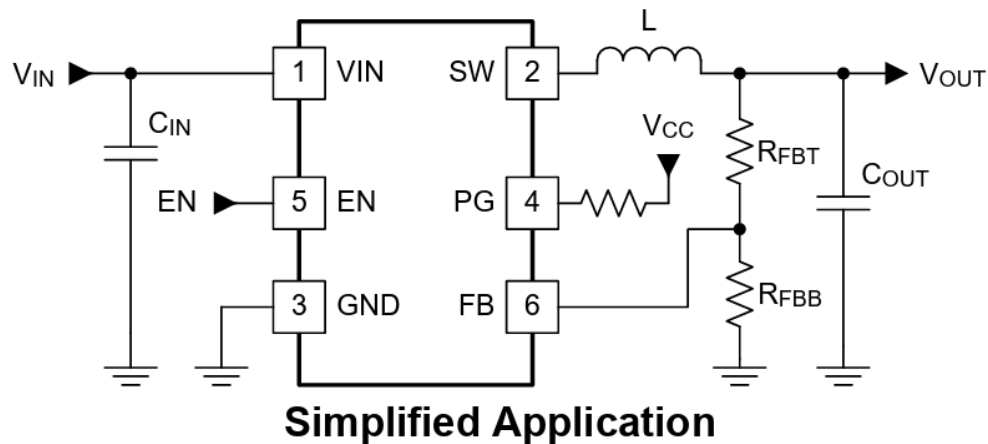


Figure 6.1.1.1

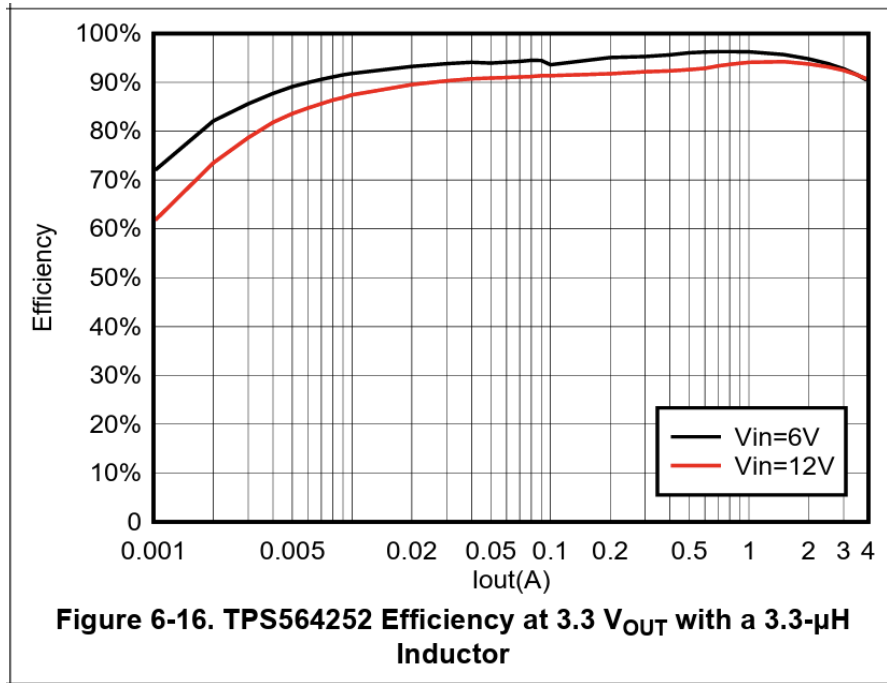


Figure 6.1.1.2

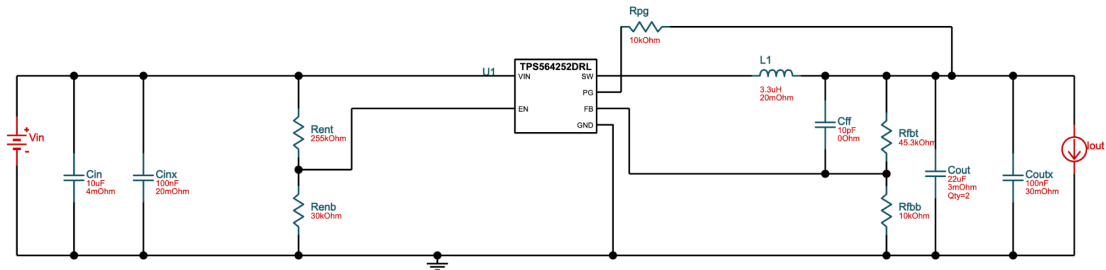


Figure 6.1.1.3

Using TI WEBENCH power designer we customized this TPS564252 voltage regulator circuit to meet our power specifications for the ESP 32. The simulated outcome and estimated BOM are as follows:

1. **Vout Actual:** 3.32 V
2. **Vout Tolerance:** 2.67%
3. **Total BOM:** \$0.72
4. **BOM Count:** 13
5. **Vout:** 3.3 V
6. **Duty Cycle:** 28.29%
7. **Efficiency:** 94.2%
8. **Frequency:** 620.82 kHz
9. **Pout:** 6.6 W
10. **Mode:** CCM
11. **Vout Peak-to-Peak:** 8.31 mV

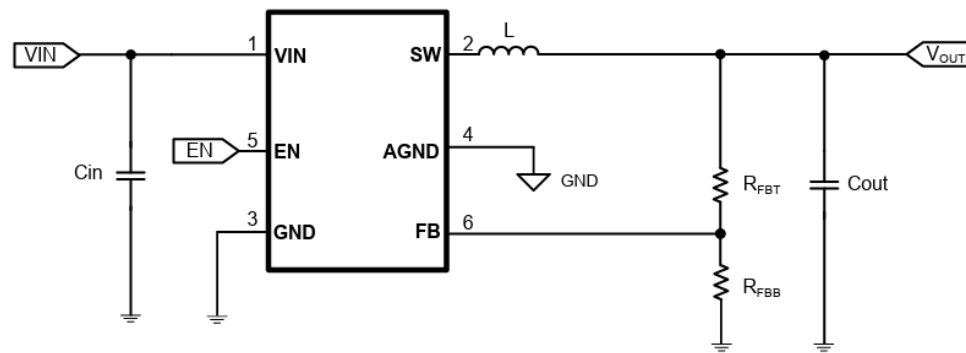
12. Vin Peak-to-Peak: 368.66 mV

13. Iout: 2A

In conclusion, the TPS564252 proves itself to be a cost effective and efficient choice as a voltage regulator for the ESP 32, performing with an efficiency rate over 90% when given a steady input voltage of around 12V (Figure 6.1.2).

6.1.2 Raspberry Pi Power Supply Regulator - TPS566242

The TPS56624x is a line of regulators that take 3V-16V inputs and can convert them to a range of low voltage outputs 0.6V - 7V at a 6-A continuous current. The TPS566242 supplies a steady 5V output with a great operational efficiency at a 12V input, making this model in the TPS line of regulators meet the specifications of our Raspberry Pi perfectly.



Simplified Schematic

Figure 6.1.2.1

Figure 6.1.2.2

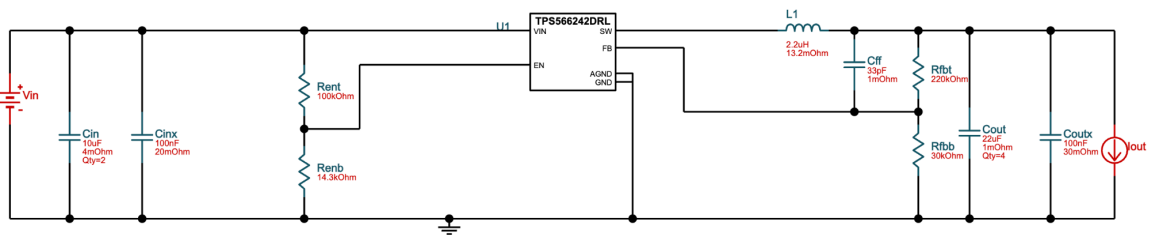


Figure 6.1.2.3

Using TI WEBENCH power designer we customized this TPS566242 voltage regulator circuit to meet our power specifications for the Raspberry Pi 5. The simulated outcome and estimated BOM are as follows:

14. Vout Actual: 5 V

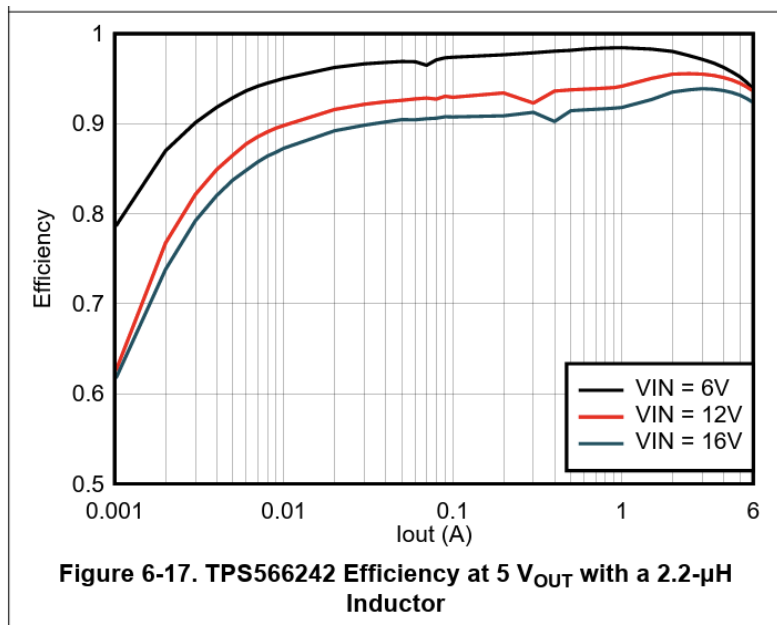
15. Vout Tolerance: 3.64%

16. Total BOM: \$1.35

17. BOM Count: 15

18. Vout: 5 V

19. Duty Cycle: 42.8%



20. Efficiency: 94%

21. Frequency: 570.11 kHz

22. Pout: 25 W

23. Mode: CCM

24. Vout Peak-to-Peak: 11.6 mV

25. Vin Peak-to-Peak: 600.86 mV

26. Iout: 5A

In conclusion, the TPS566242 proves itself to be a cost effective and efficient choice as a voltage regulator for the Raspberry Pi 5, performing with an efficiency rate over 90% when given a steady input voltage of around 12V (Figure 6.1.2.2).

7. Software Design

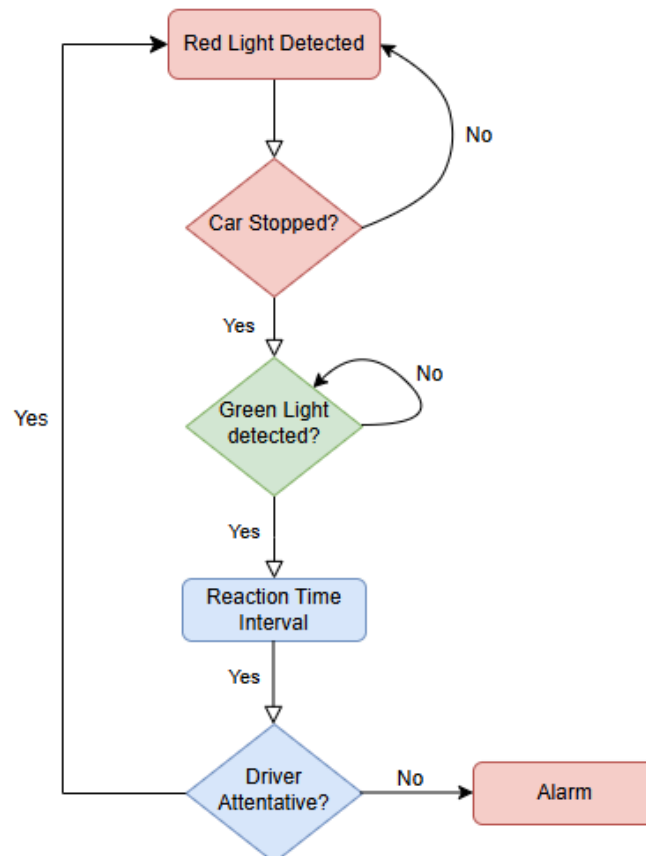
7.1 Software Logic Introduction - Scenarios

Before an approach to coding any implementation of software design, we must first consider the various scenarios our Safe Sight product will encounter on the roadways, then create ideas of software implemented logic, and finally execute and implement software to our design. This process requires careful anticipation of change and modularity to ensure the system remains adaptable and scalable as new challenges arise. Additionally, abstraction plays a critical role in isolating essential behaviors from implementation details, allowing us to focus on high-level problem-solving without being bogged down by unnecessary complexities.

7.1.1 Red-Light Scenario

The first scenario, the scenario that inspired our product, is the red-light scenario. The red light scenario consists of the operator's vehicle stopped at a red light. Having two outcomes, the driver of the vehicle housing our safe sight device will either react accordingly in a timely manner to the switching of the traffic light from red to green, or fail to react accordingly due to their attention being absent to the roadway. Now, if the driver remains attentive in the scenario, our safe sight product will do nothing, for the driver did not commit an attention infraction. However, if the driver fails to react to the changing traffic signal, Safe Sight will activate an alarming tone and take a picture of the driver, documenting the attention infraction's date, time, and record media of the incident. A logic flow graph is pictured below to demonstrate this scenario.

Figure 7.1.1.1



7.1.2 Driver Distracted While Driving Scenario

Furthermore, another scenario where Safe Sight will be implemented is in the event of a driver who is distracted while their vehicle is in motion. The driver's attention to the roadway will be assessed based on their overall head posture. If the driver is looking down, for example, texting and driving or changing the song on their stereo, they will be flagged as committing an attention infraction, and Safe Sight will alarm the driver and document the offense. This can be structured as software logic by using real-time data from cameras and sensors to track head movements and compare them against predefined thresholds for safe driving behavior. For instance, if the system detects a downward gaze lasting longer than a set duration, it triggers an alert and logs the incident in a database for future reference. However, if the driver maintains their attention to the road with minimal and normal shifts in their overall posture, Safe Sight will do nothing and remain silent. By employing machine learning algorithms, the software can also adapt over time to recognize patterns of natural head movements versus distracted behaviors, ensuring greater accuracy and fewer false positives. A logic flow graph is pictured below to demonstrate this scenario.

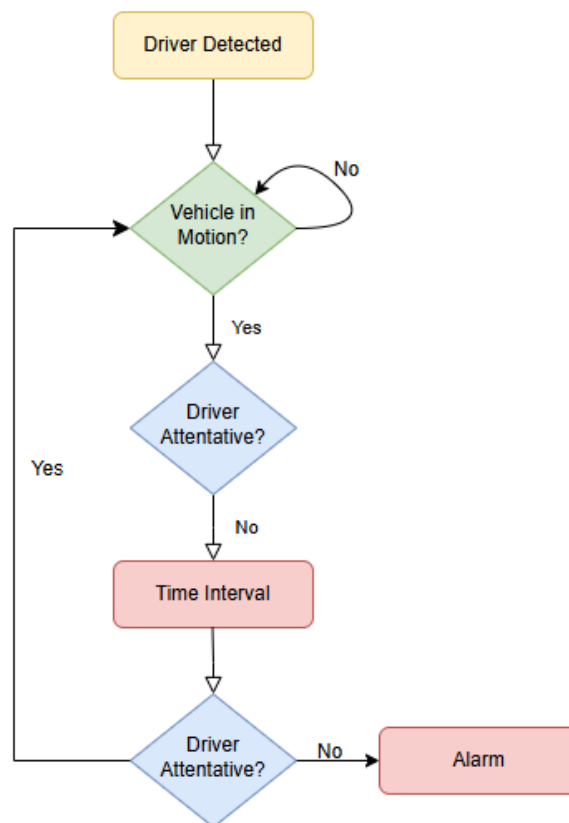


Figure 7.1.2.1

10. Administrative Content

10.1 Budget and Financing

As stated in Chapter 4, one of the biggest challenges for this project is the funding generated from senior year students not working in the industry. With that being said, initially we began with a maximum budget with the initial BOM being under \$300 which would evenly distribute the cost per member of this team to be \$100. But after some requirements to meet constraints of regulations, functionality, adjustments to the hardware, and performance benchmarks, some of the costs ended up exceeding the initial proposal of ideas. For instance, during testing we needed more ESP32 boards in order to test different subsystems in the project. This led to the cost of a microcontroller unit to be multiplied by 3 in order to rely on other microcontrollers for simultaneously testing subsystems at the same time. The cost for PCB boards will also be more than anticipated as we will adjust the

Research Links

Raspberry Pi Datasheet:

<https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf>

UART:

<https://thinkrobotics.com/blogs/learn/deep-dive-into-uart-protocol>

SPI:

<https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/>

<https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html>

I2C:

Buzzer:

<https://www.quisure.com/blog/faq/what-is-the-working-principle-of-the-buzzer>

<https://squishycircuits.com/blogs/articles/deep-dive-mechanical-vs-piezoelectric-buzzer>

AI:

<https://www.nvidia.com/en-us/glossary/tensorflow/>

Radar notes:

- <https://www.dfrobot.com/product-2793.html>
 - **RADAR OF CHOICE**
 - **25M detection through windshield**
 - **UART and ESP32 compatible**
- Why the other radars were not applicable to our project:
 - Distance detection was too short for preventative measures
 - Many radars were only functional with bigger voltage requirements
 - Many radars were absurdly expensive for budget (Banner t30R)
 - Many radars were not compatible with the ESP32 or UART
- **Links:**
 - <https://www.pepperl-fuchs.com/korea/ko/51481.htm>
 - https://bgaa.by/sites/default/files/inline-files/212-joonhyunghi-pid1908293_0.pdf
 - <https://community.infineon.com/t5/Knowledge-Base-Articles/Under-standing-FMCW-Radars-Features-and-operational-principles/ta-p/767198#>.
 - <https://ondosense.com/en/radar-know-how-optimal-use-of-radar-sensors/fmcw-radar-sensor-basics/>
 - <https://www.radartutorial.eu/02.basics/Frequency%20Modulated%20Continuous%20Wave%20Radar.en.html>
 - <https://wirelesspi.com/fmcw-radar-part-1-ranging/>
 - https://skynet.ee.ic.ac.uk/notes/Radar_6_CW_Radar.pdf
 - <https://www.keysight.com/blogs/en/tech/educ/2023/automotive-radar>
 - <http://my.avnet.com/abacus/solutions/markets/automotive-and-transportation/automotive/comfort-infotainment-and-safety/automotive-radar/>
 - <https://www.dfrobot.com/product-2793.html>
 - <https://www.lockheedmartin.com/en-us/news/features/2021/how-do-radars-work.html>

- <https://forum.arduino.cc/t/lidar-sensor-not-seeing-through-car-windshield/917780>
- <https://pmc.ncbi.nlm.nih.gov/articles/PMC10781233/>
- <https://cris.fau.de/publications/325221941/>
- <https://pmc.ncbi.nlm.nih.gov/articles/PMC10459047/>
-

Research links (Matt):

<https://pwwskills.com/blog/c-programming-language-history-invention-timeline-more/>
https://www.tutorialspoint.com/cprogramming/c_history.htm
<https://www.quickstart.com/blog/programming-language/learn-c-programming-in-10-days/>
https://www.tutorialspoint.com/swift/swift_overview.htm
<https://www.swift.org/about/>
<https://developer.apple.com/swift/>
https://www.tutorialspoint.com/java/java_overview.htm
<https://itpreneurpune.com/features-of-java/>
<https://www.bairesdev.com/blog/things-you-can-create-with-java/>
https://www.w3schools.com/python/python_intro.asp
<https://www.codecademy.com/article/learn-python-python-syntax>
https://www.tutorialspoint.com/python/python_overview.htm
https://dev.to/g_venkatasandeepreddy_b/overview-of-github-enterprise-3jno
https://ubc-library-rc.github.io/intro-git/content/04_github.html
<https://about.gitlab.com/topics/version-control/what-is-git-version-control/>
<https://en.wikipedia.org/wiki/Bitbucket>
<https://www.atlassian.com/software/bitbucket>

MCU links:

<https://www.espboards.dev/blog/esp32-alternatives/#stm32-series>
https://startingelectronics.org/articles/ESP32/esp32-introduction/?utm_source=chatgpt.com
https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/?utm_source=chatgpt.com
https://en.wikipedia.org/wiki/Raspberry_Pi#:~:text=Raspberry%20Pi%20Pico%20was%20released,2%20MB%20of%20flash%20memory.
<https://shop.pimoroni.com/products/raspberry-pi-pico?variant=32402092294227>
<https://www.raspberrypi.com/products/raspberry-pi-pico/>
<https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf>
<https://micropython.org/>
<https://circuitpython.org/>
<https://en.wikipedia.org/wiki/STM32>
https://www.st.com/content/st_com/en/stm32-mcu-developer-zone/developer-resources.html
<https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcu-s.html>
<https://www.st.com/resource/en/datasheet/stm32h743vi.pdf>
<https://www.utmel.com/components/stm32h7-arm-cortex-m7-mcus-features-advantage-and-datasheet?id=1112>
<https://hilelectronic.com/beginners-guide-to-stm32-microcontroller/>
https://m.media-amazon.com/images/I/61o2ZUzB4XL._AC_UF894,1000_QL80_.jpg
<https://cdn-shop.adafruit.com/970x728/5544-02.jpg>

https://i.kickstarter.com/assets/013/330/729/eca6f3896c292ca814915550e69f6eb5_original.png?anim=false&fit=cover&gravity=auto&height=873&origin=ugc&q=92&v=1473165725&width=1552&sig=WwfCMHfBXPwHFJlQrWKwkUOnS3Ko9CbHdKILo6miqrQ%3D
<https://m.media-amazon.com/images/I/61R1A7CuHTL.jpg>
<https://docs.arduino.cc/resources/datasheets/A000005-datasheet.pdf>
<https://docs.arduino.cc/hardware/nano/#tech-specs>
<https://www.ettechnophiles.com/introduction-to-atmega328p-pinout-datasheet-specifications/>
<https://www.alldatasheet.com/datasheet-pdf/download/1425005/MICROCHIP/ATMEGA328P.html>
<https://www.seeedstudio.com/blog/2019/11/07/arduino-communication-peripherals-uart-i2c-and-spi/>
<https://www.kevsrobots.com/resources/boards/teeny4-1.html>
<https://www.joy-it.net/en/products/Teensy40>

Standards:

<https://jlcpcb.com/capabilities/pcb-capabilities>
<https://www.protoexpress.com/blog/ipc-2221-circuit-board-design/>
<https://www.mclpcb.com/blog/ipc-standards-for-pcbs/>
<https://www.ipc.org/TOC/IPC-2221A.pdf>
IPC. *IPC-2221C: Generic Standard on Printed Board Design*. IPC International, 2023.
https://resources.pcb.cadence.com/blog/applying-ipc-pcb-design-standards-cadence?utm_source=chatgpt.com

Understanding UART: Serial Communication for Embedded Systems. Texas Instruments, 2015, www.ti.com.

"UART – Universal Asynchronous Receiver Transmitter." *Electronics Tutorials*, www.electronics-tutorials.ws/communication/uart.html

EIA/TIA-232-F. *RS-232 Interface Standard*. Electronic Industries Alliance, 1997.

EIA-485. *RS-485 Standard*. Electronic Industries Alliance, 1998

<https://resources.altium.com/p/serial-communications-protocols-part-two-uart>

<https://www.analog.com/en/resources/technical-articles/fundamentals-of-rs232-serial-communications.html>

<https://barenakedembedded.com/uart-vs-rs232/>

<https://www.optcore.net/difference-between-rs-232-rs-422-and-rs-485/>

UM10204: I²C-bus Specification and User Manual. Rev. 6, NXP Semiconductors, 2014, www.nxp.com/docs/en/user-guide/UM10204.pdf

Basics of the I²C Communication Protocol. Texas Instruments, 2015, www.ti.com/lit/an/slva704/slva704.pdf

Understanding the I²C Bus. Analog Devices,

www.analog.com/en/technical-articles/understanding-the-i2c-bus.html

https://www.researchgate.net/figure/Multiple-master-and-slave-connection-in-I2C_fig1_281953226

INERTIAL SENSOR LINKS

<https://joy-it.net/en/products/SEN-MPU6050>

<https://iies.in/blog/mpu-6050-unraveling-the-wonders-of-a-6-dof-inertial-measurement-unit/>

<https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>

<https://www.elprocus.com/mpu6050-pin-diagram-circuit-and-applications/>

<http://maker.pro/arduino/tutorial/how-to-interface-arduino-and-the-mpu-6050-sensor>

<https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/>
<https://randomnerdtutorials.com/arduino-mpu-6050-accelerometer-gyroscope/>
<https://learn.adafruit.com/mpu6050-6-dof-accelerometer-and-gyro/pinouts>
<https://components101.com/sensors/mpu6050-module>
<https://circuitdigest.com/microcontroller-projects/interfacing-mpu6050-module-with-arduino>
<https://www.instructables.com/Accelerometer-MPU-6050-Communication-With-AVR-MCU/>
<https://forum.arduino.cc/t/getting-started-with-the-mpu6050-beginner-friendly-guide-with-visual-explanations/1364064>