Hazardous Air Vehicle

EE 475

Sam Johnson

Emraj Sidhu

Table of Contents

ABSTRACT

Amongst the common public, wanting a vehicle which can handle hostile environments is a key criteria for their selection. Compounding this desire is the slow degradation of the environment which has led to harsher weather conditions throughout the year in many regions. An example of this seen in New Delhi last November, when the capital was paralyzed due to smog caused by air pollution. This resulted in terrible breathing problems if one stepped outside. Our aim via this project is to introduce an exploration vehicle which can handle such conditions and provide the user vital information about the environment through the use of air sensors and a live video feed.

INTRODUCTION

This project is aimed at introducing an exploration car which can handle such harsh conditions, especially in relation to the consequences of air pollution (experienced near major industrial towns). The first function of the designed vehicle is to detect the amount of air pollutants in the environment around the use, namely carbon dioxide (CO2) and total volatile organic compounds (TVOC); alerting him/her as to when levels are extremely dangerous, unsafe, or completely safe. The second function is to provide a live video feed to a webpage. This video feed will come from the camera attached on the vehicle. The end result is that the user will not only be alerted when entering areas with terrible air quality but also be able to proceed safely in case it is needed. Additional functionality includes the measuring (and display) of temperature and humidity in the environment around the vehicle as well as the ability to control the position of the camera.

DESIGN SPECIFICATION (Software/Hardware)

Design Specification

System Description

This specification describes the design requirements for a prototype of a remotely controlled, short-range, hazardous air vehicle. To test the feasibility of this project, the prototype system will implement three main system components: the vehicle component, with the main control implemented on a Raspberry Pi, the bluetooth remote component, implemented on a PIC1825K22 microcontroller, and a website for information display to the user. The user will control the system from the remote, sending information over bluetooth to the vehicle to control motor speed and direction. This will allow the user to navigate through an environment by controlling the motion of the vehicle. The website will be accessible over wifi, allowing the user to view live video from the vehicle as well as air quality information about the surrounding environment. The website will notify the user of any hazardous levels of pollutants in the air that may be dangerous to the user. In addition to this, the webpage will also allow the user to change the position of the camera (pan and tilt functionality) through the use of a servo arm.

Specification of External Environment

The prototype system is a proof of concept for control, communication and usability. As a result, the system must operate in a controlled laboratory environment with minimal outside temperature or other conditions to negatively affect operation. The system will be powered by on-board power sources to prove the mobility of the vehicle.

Temperature range : 15 - 30C
Humidity: 40-50%
Weather: Lab Environment (Dry, no precipitation)
Power Supply: 5V (logic) and 6V (motor)

System Input and Output Specification

System Inputs

User Input - The system will support user input to control the motion of the vehicle from the bluetooth remote component. This controller will be tilt-based, and control the motion of the vehicle based on the orientation of the sensor (accelerometer) with respect to the gravity vector. This sensor will be set to have the following input boundaries:

- Sensing Directions: x, y, z
- Dynamic Range: +-4g
- Data Range: 14 bits per direction
- Sensor Read Range: -8192 to 8191 (corresponding to -4 * g to 4 * g where g is 9.8 m/s^2)
- The sensor will use the gravity vector for tilt, so the max accepted input is -2048 to 2047

Air Quality - The system will support measurement of the surrounding air quality, based on the amount of CO2 and TVOC (total voltatile organic compounds) present in the environment. This input will be onboard the vehicle component of the project. The input will be two readings:

- Equivalent CO2 reading
    - Measured from 0 - 60000 parts per million
- TVOC reading
    - Measured from 0 - 60000 parts per billion

Live Video - The system will support live video capture onboard the vehicle component. The video will have the following specifications:

- Quality: 720P
- Frame Rate: 30 fps
- Codec: H.264
- Format: mp4

Temperature/Humidity - The system will support temperature and relative humidity measurements. The measurements will be able to measure the following temperature range.

- Temperature range: -10 to 85 Celsius
- Error: +- 0.4C
- Humidity Range: 0 to 80% RH
- Error: +-3%

Pan/Tilt - The system will allow the user to change the angle at which the camera is positioned. Specifically, the user will be allowed to "pan" the camera (turn it right or left) and tilt it (turn it up or down). Initially the camera will be positioned at 90 degrees tilt and 90 degrees pan.

- Range for pan/tilt - Between 30 degrees and 150 degrees.
- Increment for pan/tilt: +10%
- Decrement for pan/tilt: -10%

System Outputs

User Output (Website Display) - The system will support feedback to the user about the surrounding environment via a website. The website will display the following information:

- Live Video: 720p30 from vehicle component
- eCO2 and TVOC readings from the vehicle component
- Temperature and Humidity readings from the vehicle component
- The position of the camera in degrees (pan and tilt)

User Interface

The user will be able to control vehicle via the bluetooth connected tilt-based remote control. The remote control will allow the user to smoothly control the movement based on the orientation of the remote, and will allow the user to execute the following movements with the vehicle. Each movement will have a range of speeds, based on the magnitude of the tilt that the user applies to the controller.

- Drive Forward (0 - 0.5 m/s)
    - Corresponds to 6 - 24 degrees of tilt forward
- Drive Backward (0 - 0.5 m/s)
    - Corresponds to 6 - 24 degrees tilt backwards
- Turn Right/ Left while moving forward (0 - 90 degrees/second, 0.05 m/s)
    - Corresponds to 6 - 24 degrees of tilt forward
    - 9 - 90 degrees left/right
- Turn Left on the spot (0 - 90 degrees/second, 0.05 m/s)
    - Corresponds to 6 - 24 degrees of tilt backward
    - 9 - 90 degrees left/right

The feedback from the system will be displayed to the user on a convenient website, that is accessible via wifi. The user interface will display convenient

information from the vehicle to allow the user to judge the air quality of the surrounding environment. The website will also allow the user to change the position of the camera. To display this appropriately, the website will display the information to the user, described in the table below.

Table 1: Website Information

Data	Description
720p30 Live Video	This video will allow the user to view the surrounding environment visually, The user can then detect whether the air is smoggy, foggy, etc and can navigate around obstacles.
eCO2 reading	The CO2 concentration in the air, measured in 0-60,000 ppm (parts per million)
TVOC reading	The total volatile organic compound concentration in the air, measured in 0-60,000 ppb (parts per billion)
Temperature reading	The temperature in the environment surrounding the vehicle measured in C.
Relative Humidity reading	The relative humidity reading in the environment surrounding the vehicle measured in %.
Position of Camera (pan and tilt)	The position of the camera will initially be set at 90 degrees for pan and tilt. The user can increment or decrement either position by 10 degrees.

Additionally, to make the information readily understandable for the user, the eCO2 reading will be accompanied with warning level indicators to notify the user of danger levels. The warning levels are based off of the information found at https://www.kane.co.uk/knowledge-centre/what-are-safe-levels-of-co-and-co2-in-rooms , and will range in warning level from 0 - 4, described in the table below.

Table 2: CO2 Concentration level warnings

Warning Level	CO2 concentration	Harm Level
0	0 - 1000 ppm	None: normal Levels
1 (Green)	1000 - 2000 ppm	Low: stuffy air

2 (Orange)	2000 - 5000 ppm	Moderate: short term hinderance
3 (Pink)	5000 - 40000 ppm	Severe: highly dangerous
4 (Red)	> 40000 ppm	Extremely Severe: permanent damage

The warning level will be displayed below the concentration reading for eCO2. If the warning level reaches level 2 or higher, the user will be notified with a pop-up warning on the display of the user interface with the appropriate warning level to notify the user. This is to ensure that the danger level of the environment is appropriately communicated to the user.

The TVOC reading will also be accompanied by warning level indicators to inform the user of potential threats to health. The warning levels are based on the information found at
http://www.critical-environment.com/blog/know-the-air-you%E2%80%99re-breathing-volatile-organic-compound-2-of-4/
Since the system will measure TVOC concentrations, and not each individual concentration, it is hard to specify fool-proof warning levels (since different chemicals have different harmful thresholds). To combat this, the following warnings will error on the safe side of warning the user. However, ultimately it will be the user's discretion to judge the reading based on what VOCs might be present in the given environment. The warning levels will range from 0 - 2 and are described in the table below.

Table 3: TVOC concentration level warnings

Warning Level	CO2 concentration	Harm Level
0	0 - 7000 ppb	Low: low levels of concern
1 (Orange)	7000 - 20000 ppb	Moderate:
2 (Red)	>20000 ppb	High

System Functional Decomposition

This system is intended to prototype a hazardous air vehicle, which can be controlled by the user and collect data on the CO2 concentrations in the surrounding environment in order to analyze the air quality. This functionality can be divided into three parts: the actual vehicle with the raspberry pi embedded within it, the bluetooth remote component, implemented on a PIC1825K22 microcontroller, and a website for information display to the user.

The raspberry pi will be the “heart” of the system connected to the vehicle and the CO2 sensors. It will be connected to the PIC via bluetooth and will serve up the website via wi-fi, with the corresponding modules for these items set-up. It will run the software needed to control the direction of the vehicle through the h-bridge. Additionally, it will also stream video (from the camera connected to it) to the website. Lastly, it will also be wired to the CO2 sensors and collect/analyze the data being given by them.

The bluetooth remote component will be implemented using the UART of the PIC1825K22 microcontroller in conjunction with the blueSMiRF bluetooth module. An accelerometer will be wired to the PIC so that the controller is able to sense its orientation with respect to the gravity vector. The microcontroller will also be connected to the blueSMiRF bluetooth module, which in turn will connect with the raspberry pi (located on the vehicle) through bluetooth. This will enable the user to control the vehicle and steer it in whatever direction he/she wants.

The website will created through Python scripts on the Pi. It’s main objective is to stream video from the camera connected to the raspberry pi as well as allow the user to control the position of the camera. Alongside with this, the website will also display information collected from the sensors. The user will able to see warning messages depending upon the amount of CO2 pollution outside the vehicle.

System Architecture

Furthermore, the system breaks down into smaller functional blocks shown in the block diagram below.

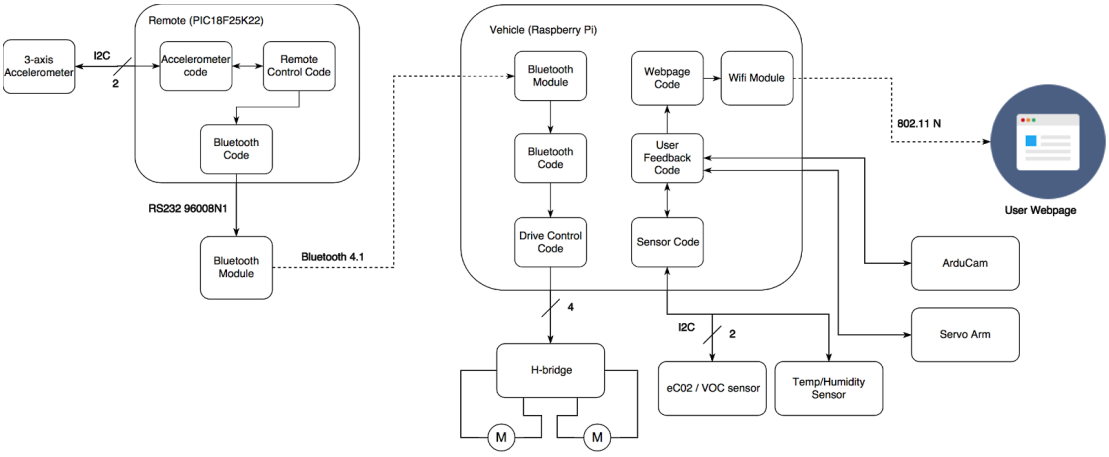

Figure 1: Block diagram of overall system

Each of these blocks and their function is briefly described below.

3 Axis Accelerometer - Senses acceleration in the x, y and z directions to detect the orientation of the bluetooth remote with respect to the gravity vector. This sensor will be accessed via the following bus:

- I2C @ 100kHz

Accelerometer Code - This interface allows other code on the remote to configure the accelerometer and read sensor information from it.

Remote Control Code - This is the main control code on the remote. It's purpose is to read orientation information from the accelerometer, format it properly, and then send it to the vehicle over the bluetooth communication channel.

Bluetooth Code (Remote) - This interface allows the remote control code to interface with the bluetooth module to send commands to the vehicle.

Bluetooth Module (Remote) - This module allows the remote to send information via bluetooth. This module will be accessed from the remote via the following interface:

- RS232 9600 8N1

Bluetooth Module (Vehicle) - Build in bluetooth module on the Raspberry Pi for communication with the remote. This module will be linked via bluetooth 4.1 with the bluetooth module on the remote.

Bluetooth Code (Vehicle) - This code will be responsible for capturing and storing the incoming commands to the vehicle and notifying the control code when new information has been received.

Drive Control Code - This code will control the speed and direction of the motors based upon the accelerometer orientation information received over bluetooth from the remote. It will then control the motor speed and direction based upon this information, using two GPIOs and two PWMs connected to an H-bridge module.

H-Bridge - This module allows the two connected DC motors to be controlled in terms of speed and direction of rotation.

eCO2 / VOC sensor - Senses the eCO2 (effective CO2) and TVOC (total volatile organic compound) levels in the surrounding air. The sensor information will be accessed via the following interface:

- I2C @ 100kHz

Temperature/Humidity sensor - Measures the surrounding temperature and humidity around the vehicle. The sensor information will be accessed via the following interface:

- I2C @ 100kHz

Sensor code - This code is responsible for periodically reading the air quality sensor values and storing them for the user feedback to read and process.

ArduCam - This module allows video capture on the vehicle with the following specifications:

- 720p resolution
- 30 fps frame rate
- H.264 codec

Servo Arm for camera - This module allows the camera pan and tilt angles to be changed in order adjust the camera view on the vehicle.

User Feedback Code - This code is the main code running on the vehicle to send live video and sensor information over wifi to the android application.

Webpage Code - This code takes the live video feed, camera position, and the sensor information and creates a web server to display the information to the user over wifi.

Wifi Module - This is the on-board wifi module, capable of the following protocol:

- 802.11 N wifi

User Website - This website takes the live video feed, camera position, and sensor information from the wifi connection, and displays it in a manner that is understandable by the user.

Operating Specifications

As a prototype the system will operate in a controlled laboratory environment.

Temperature: 15-30C

Power: Various Batteries

Reliability and Safety Specification

The system will be designed to be as safe and reliable in its purpose as possible, issuing warnings to the user if the surrounding air is hazardous to human health. It will run off of on-board power supplies, and will be as reliable as possible. Since the system will operate in the U.S. , it will comply with the following specifications and regulations:

- 47 CFR 15: Code of federal regulations for unlicensed transmissions
- 802.11 b/g/n Wireless LAN

- Bluetooth 4.1
- 15 U.S.C. 1278a

DESIGN PROCEDURE

Developing this project was divided into two large portions. One portion was the drive-control system, and the other portion was the development of the monitoring system. Each portion consisted of many different components, and was developed separately from the other. These two large portions of the project were integrated at the end to complete the development of the entire project.

Drive - Control

Bluetooth Remote

The design of the drive-control system began with the development of the bluetooth remote. The first step in the development process was interface the accelerometer for the tilt-based remote with the PIC1825K22 microcontroller. This involved writing a library for the PIC to reset, read and write to the accelerometer. Once all the errors with the I2C protocol between the accelerometer and the PIC were corrected, the next step in the design process was to interface the PIC with the bluetooth module. This involved implementing the correct UART communication protocol on the PIC to send information to the bluetooth module. This step also involved properly configuring the bluetooth module into slave mode for pairing with the vehicle.

Next, the power for the bluetooth remote was implemented using the 5V power lines of the USB interface. The 5V lines were used to power the PIC, the accelerometer and the bluetooth module onboard the remote.

Vehicle Drive System

The design of the drive system began with the pairing of the bluetooth module on the Raspberry Pi to the bluetooth module on the tilt-based remote. This included setting up a serial interface in the Linux OS, and reading the tilt information from the bluetooth stream using Python. Once this connection was established, the H-bridge was connected to the Raspberry Pi using the GPIO bank. Then, a control algorithm was implemented in Python to convert the tilt information from bluetooth into control for the motors connected to the H-bridge.

Monitoring System

Sensors

The interface with the sensors began by making the connection with the I2C pins on the Raspberry Pi to the I2C connections on the temperature/humidity and TVOC/CO2 sensors. The Linux OS was then configured to enable I2C communication. Once the communication was established, libraries were written to reset and read the sensor data over I2C. These libraries were packaged into a class to be able to easily setup the sensors and read data back from them.

Webpage

The webpage was created through the use of HTML, javascript, and python files. When the user clicked a button to increment or decrement the pan or tilt angles, the HTML code would link to a python file containing software to control the servo arm (which the camera is attached to) through the use of an <a href> tag. The overall python file (called app.py) used to control the webpage would then import the servo control file and continually send the updated position of the camera back to the webpage. The video feed was established through existing software given by Raspberry Pi. It created a camera object and frame by frame motion. This file was also imported into app.py in order to send the video feed to the webpage URL.

SYSTEM DESCRIPTION

On a high level, the hazardous air vehicle consists of the vehicle portion and the bluetooth remote portion. The bluetooth remote portion is controlled using a PIC18F25K22 microcontroller. It's main purpose is to read tilt-information from the 3-axis accelerometer over I2C and to continuously send the information over bluetooth to the vehicle in order to control the movement of the vehicle. The vehicle portion, controlled by the Raspberry Pi, then receives these commands, processes them and produces the appropriate control signals to control the motors connected to the H-bridge. The vehicle portion is also responsible for hosting the webpage over WiFi for the user to view the live video feed, and the feedback information from the TVOC/CO2 and Humidity/Temperature sensors, and to control the viewing angle of the camera. The overall block diagram of the system is found in the figure below.

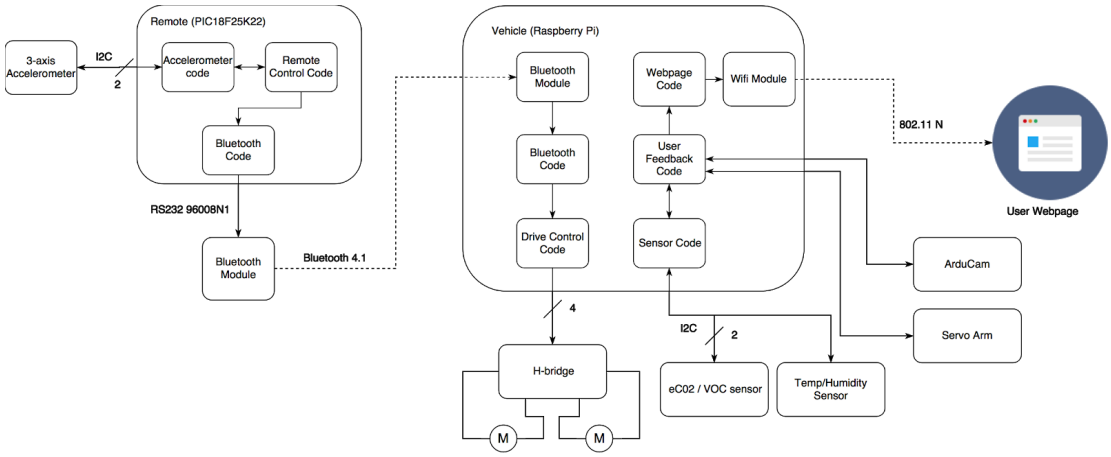

Figure 2: Block diagram of overall system

Bluetooth Remote

Inputs

The input to the bluetooth remote is the orientation of the remote with respect to gravity. The remote reads in this information by reading the information from the 3-axis accelerometer in the X, Y, and Z directions. The positive Y direction is forward, the

positive Z direction is down, and the positive X direction is right. The acceptable input range for the sensor is described in the table below.

Table 4: Input range for Bluetooth Remote

Direction	Acceleration Range	Data Range
X	+-4g	-8096 to 8095
Y	+-4g	-8096 to 8095
Z	+-4g	-8096 to 8095

Output

The output of the bluetooth remote is a continuous stream of tilt information sent via bluetooth. This consists of the tilt data read from the accelerometer at 2kHz formatted as described below:

- 8 byte sequence
- 2 bytes for x data, 2 bytes for y data, 2 bytes for z data, '/r/n' characters

Drive Control

Input

The input of the drive control system is the same as the output of the bluetooth remote. Due to the rate at which the data is being sent, about 1 in every 20,000 receives is corrupted in terms of the number of bytes received. These erroneous samples of data are ignored.

Output

The output of the drive control system is the control signals to the H-bridge which control the direction and rotation speed of the two DC motors. This consists of 6 output pins: four GPIOs to control the direction of rotation and 2 software PWMs to control the rotation speed.

Webpage

Input

The input into the webpage is the user option to change the position of the camera. This is done by incrementing or decrementing the pan and/or tilt position by 10 degrees.

Output

The output of the webpage is the video feed from the camera attached to the vehicle, the current camera position (in degrees), and the data from the air quality and temperature sensors along with the associated warning level.

SOFTWARE IMPLEMENTATION

The software for this system was developed using MCC developed code and adding significant modifications on top of it as well as Python and other libraries. Since there are many

software components, this section will be broken down into smaller sections describing each component. Each of these systems was tested independently to ensure proper functionality before full integration with the system. For the overall view of the system please refer to the block diagram in the system description section of this report.

Bluetooth Remote

Accelerometer Code

The accelerometer code is a library for controlling the MMA8451 3-axis accelerometer. It allows the user to initialize the sensor, and provides functions for reading and writing specific registers. This library is implemented on top of the MCC generated I2C library code that has been heavily modified for this specific sensor. Specifically, the MCC generated library only provides the most basic read and write sequences, so the I2C code was modified to accommodate the specific read and write format that this sensor requires for operation. The table below shows the functions that were implemented in the library to communicate with the accelerometer.

Table 5: Accelerometer Library Functions

Function Call	Description
void ACCELEROMETER_Initialize(void)	Initializes the accelerometer by resetting the sensor. Additionally, checks that the device is the correct sensor by checking some user registers.
void write_register(uint8_t reg, uint8_t data)	Write a value to the given register.
uint8_t read_register(uint8_t reg)	Read a value from the given register.
accelerometer_data_t read_xyzvalues(void)	Read the XYZ vectors from the sensor.

For more information on the library, please refer to the code files *accelerometer.h* and *accelerometer.c* located in our code under /RemoteFinal.X. For more information on the Sensor and the I2C interface (operating at 100kHz) please refer to the hardware implementation section.

Remote Control Code

The purpose of this code is to initialize the components on the bluetooth remote, and then provide the main functionality of the bluetooth remote. To do this, the code initializes the appropriate interfaces (I2C and UART) and then enters an infinite polling loop. In this loop it repeatedly calls the function read_xyzvalues() to get the tilt information, and then sends this over to the UART interface which is connected to the bluetooth module. With the delays in the code, this sending of information executes at approximately 2kHz. The tilt data is sent over to the vehicle component formatted as 8 bytes of data. This

formatting is described in the table below.

Table 6: Tilt value formatting for Bluetooth communication

2 bytes	2 bytes	2 bytes	2 bytes
X tilt	Y tilt	Z tilt	‘/r/n’

For more information on the library, please refer to the code files *main.c* located in our code under /RemoteFinal.X.

Bluetooth Code

The bluetooth code consists of the code that is used to communicate via UART with the bluetooth module on the bluetooth remote. This consists of the MCC generated code for EUSART 1 operating at 115200 8N1. This library is used to send information from the PIC over bluetooth to the Raspberry Pi.

For more information on the library, please refer to the code files *eusart1.h* and *eusart1.c* located in our code under /RemoteFinal.X.

Drive Control

Bluetooth Code/ Drive Control Code

The bluetooth code and drive control code is the Python code on the Raspberry Pi which reads tilt data sent from the remote over bluetooth and interprets it into the movement of the vehicle. To do this, it initializes the necessary GPIO pins on the Raspberry Pi (4 GPIOS and 2 software PWMs) to control the motors through the H-bridge. Once these pins are initialized, the module enters an infinite loops that continuously polls the serial Bluetooth interface created in the Linux OS. On every iteration, the program receives a new set of tilt values and directly converts these to the movement of the vehicle.

The tilt values are converted to a movement of the vehicle in the following way. First, the Y tilt value is checked to see if the remote is tilted forward or backward. If abs(Y) > 200, then the tilt is considered significant enough to start the vehicle moving. This corresponds to a tilt angle of about 6 degrees forward or backward. At 24 degrees of tilt forward or backward, the wheels are turning at their maximum speed. Thus, the vehicle has a range of speeds depending on the tilt of the controller.

Next, the X tilt value is used to determine the turn direction if there is any. This is done by calculating a “turn factor” to decrease the speed of the wheel in which the remote is tilted. For example, if the remote is tilted forward and left, both wheels will be set to a certain speed based upon the degree of tilt in the forward direction. Then, the left wheel will be slowed down by a factor dependent on the degree to which the controller is tilted left. This makes the vehicle turn to the left, since the right wheel is now spinning faster than the left wheel. Additionally, this allows the vehicle to turn at varying speeds dependent on the tilt of the controller.

Note, that the vehicle will only turn if the remote is simultaneously turned left or right whilst the remote is also tilted forward or backward. If the controller is merely tilted left or right, the vehicle will do nothing since there is no forward or backward component to the tilt.

The format for the tilt values being sent over the bluetooth connection is described in the software implementation section for the bluetooth remote. The data has been specifically formatted and sent in this way so that the drive control code can easily read it from the interface. By following each tilt data sequence with the two byes '/r/n' it is possible for this code to continually call ser.readline() and always get the newest set of tilt values from the interface. If for some reason there is no new tilt value to read, this operation hangs until a new line is able to be read (which means that there is new tilt available). In this way, the code to control the motors is always able to identify and receive the latest tilt data that is streaming from the remote.

With the rate at which the data is being sent, it has been observed that about 1 in every 20000 data transmissions ends up erroneous and not all 8 bytes are transmitted correctly. This results in an error on the unpack() command which if left alone results in the program crashing. To avoid this, if the data received is of an incorrect format, it is simply thrown out. This is not an issue, since the new data is almost immediately available to the system.

For more information on the library, please refer to the code files *drive.py* located in our code under /VehiclePython.

Webpage

Sensor Code (sensors.py)

The sensor code is a class called AirQualityTempSensor for controlling the SI7021 (temp/humidity) and SGP30 (TVOC/CO2) sensors over I2C. It allows the user to initialize the sensors, and provides functions for reading information to the two sensors.

This class is meant to provide an easy API for the user to get data from the sensors without having to delve into anything related to the I2C protocol. To do this, the class only exposes four functions to the user. These functions are listed in the table below.

Table 7: AirQualityTempSensor class public functions

Function	Description
getCO2	Returns the most recent CO2 reading in ppm
getVOC	Returns the most recent TVOC reading in ppb
getTemp	Returns the most recent temperature reading in celsius

getHumid	Returns the most recent relative humidity reading in %

Inside the class, there are several functionalities that are obscured from the user. First, when the class is declared it resets both sensors and checks for their existence by reading and writing specific user registers over the I2C interface. This is accomplished using the smbus library in Python that can communicate with the I2C pins. If the sensors cannot be found, the object will fail to initialize and an error will be thrown to notify the user. This is to ensure that data is being read from the appropriate sensor that is actually connected to the Raspberry Pi.

Once the sensors have been initialized, the class initializes a thread that runs in the background. This thread continuously polls the sensors at 1Hz for new data. This is done because the SGP30's measurement algorithm is most accurate when it is polled for data at 1Hz. This means that the user is able to get new data using the public methods as many times a second as they want. But the values will only update at 1Hz in the background. This abstracts the sensor level details from the end user and makes the class much easier to use.

It is important to note that the sensor values for TVOC and CO2 will not be accurate for the first 15 seconds after initialization due to hardware limitations with the sensor.

For more information on the library, please refer to the code files *sensors.py* located in our code under /VehiclePython.

Servo to Control Camera Position code (controlServo.py)
The core of this program utilizes one main function, which is *setServoAngle()*.This function will take the servo desired by the user to change position (either the "pan" servo or the "tilt" servo) and the angle at which the camera should be positioned at. The input angle is converted to an equivalent duty cycle, which means the file requires the PWM to be imported (part of the RPi library). This conversion is specified by the equation duty cycle = (angle / 18) + 3. So in setting the tilt and pan angle to 90 degrees (which is what is done in this system), the duty cycle will be set equal to 8. The system then waits 0.3 seconds to make the turn before stopping the PWM.

The last step includes sending information about the camera position to the web browser through the use of `if __name__ == '__main__':`.

Note: In our system, the range of movement for the camera for both the "pan" and "tilt" servo is limited between 30 degrees and 150 degrees.

Webpage/User Feedback Code (camera_pi.py)
This program implements the camera class needed to connect to the camera hardware and download live video frames from it (using jpg streaming). Whenever the client first connects to the server, a background thread will be running which will capture the video

frames from the Raspberry Pi. This specific program will capture frames one by one, resetting the stream after each frame is captured.

Webpage/User Feedback Code (app.py)
This program is the brains of the webpage. It's main objective is to use Flask by loading onto the python script and creating a Flask object (called 'app'). In sending information to the web browser, this program utilizes 'app.route (in conjunction with the HTML file). Under this, the program will return a Response object initialized with a specific function and then send the results to the client. For example in streaming video to the webpage, the system will come across `img src="{{ url_for('video_feed') }}` in the HTML file. It will then go the python script *app.py*, where under 'app.route('/video_feed')', a Response object initializing the camera class from *camera_pi.py* will be returned and the video will stream to the given URL.

The similar idea can be applied to incrementing or decrementing the pan or tilt servo. In the HTML file, the <href> tag will be used to link a destination. For example, when the user wants to click increment the PAN angle by 10 degrees, he/she will click on "+" in the PAN angle row on the web page. The software will then redirect to *app.py*, where it will look for `@app.route("/<servo>/<angle>")`. Then the *angleServoCtrl.py* logic will run as it has the function *setServoAngle()* which accepts a "servo" and "angle" argument. In this case, the "servo" will be the pan servo connected to GPIO 27, and the angle will be 100 degrees (90 degrees is the initial angle and adding 10 to it will be make the new angle 100 degrees). The program *angleServoCtrl.py* will then calculate the equivalent duty cycle, which *app.py* will then update and display on the webpage. The logic used to display the data from the sensors is the same, but in this case, the class *AirQualityTempSensor()* will be instantiated directly in *app.py* under `if __name__ == '__main__'`: and then the associated get methods used for the temperature, humidity, CO2, and TVOC values can be directly called.

Webpage/User Feedback Code (index.html)
This is a very simple html file used to set up the webpage. The most important feature within this program is the `<h3><img src="{{ url_for('video_feed') }}></h3>` which will directly feed the video stream from the camera to the web page. As talked about earlier, 'video_feed' will directly link to *app.py*, which will then stream to the webpage frame by frame.

Another important feature within this html file is that it will generate a hyperlink which will be passed to the web server app (*app.py*). For example
`<a href="/pan/-"class="button">-</a>` represents a simple HTML hyperlink TAG styled as a button. When the user clicks on this link, a GET /<servo>/<Decrement angle> is generated, where <servo> is the pan servo and the <decrement angle> represents the '-' symbol on the webpage needed to decrease the pan servo angle by 10 degrees. These parameters are passed to *app.py*, which will also pass on the parameters to the *angleServoCtrl.py* program to get the PWM duty cycle, decrement the given angle, have that value converted to a duty cycle (by again going to *angleServoCtrl..py)*, and send the

updated angle value to the webpage.

Finally, the <scrip></script> section of the html file is especially important, as it allows the webpage to continuously poll the backend code running in *app.py* to send it new sensor values. Once the webpage loads, this section of JavaScript repeatedly calls the getSensor() function. This function then sends an XMLHTTP request for a JSON object to the url /temp_feed on the website. Flask then catches this request and triggers the function temp_feed() in *app.py* which returns a JSON object containing all of the newest sensor readings (which are read from the class AirQualityTempSensor via its public methods). This object is then returned to the webpage where it is unpacked by JavaScript and the elements in the webpage are updated. If there are any warnings, the background of the corresponding element is changed, and a warning message is printed to notify the user.

Webpage/User Feedback Code (style.css)
This basic file is simply used for presentation purposes in relation to the webpage. It specifies the size, color and font for the buttons (used to increment the pan or tilt angles of the camera) while also doing the same for the "body" of the webpage by setting the background color and aligning the text to the center. It is used in the index.html file for style purposes.

HARDWARE IMPLEMENTATION

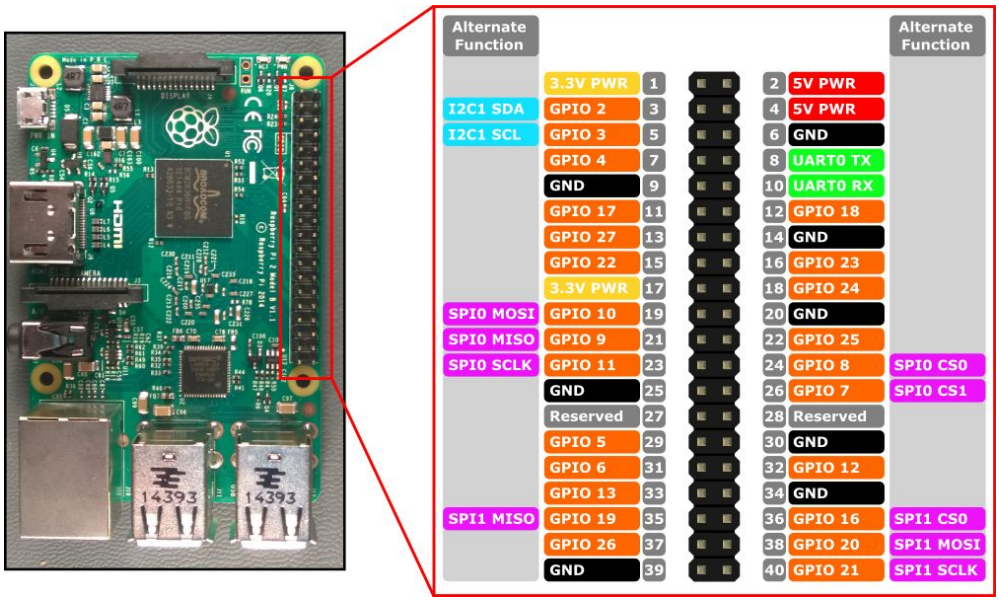

Figure 3: The pin mappings for the Raspberry Pi 3.

Bluetooth Remote

The bluetooth remote consisted of 3 main hardware components powered by the 5V power lines of the USB interface. A block diagram for the overall hardware layout is found in the figure below.

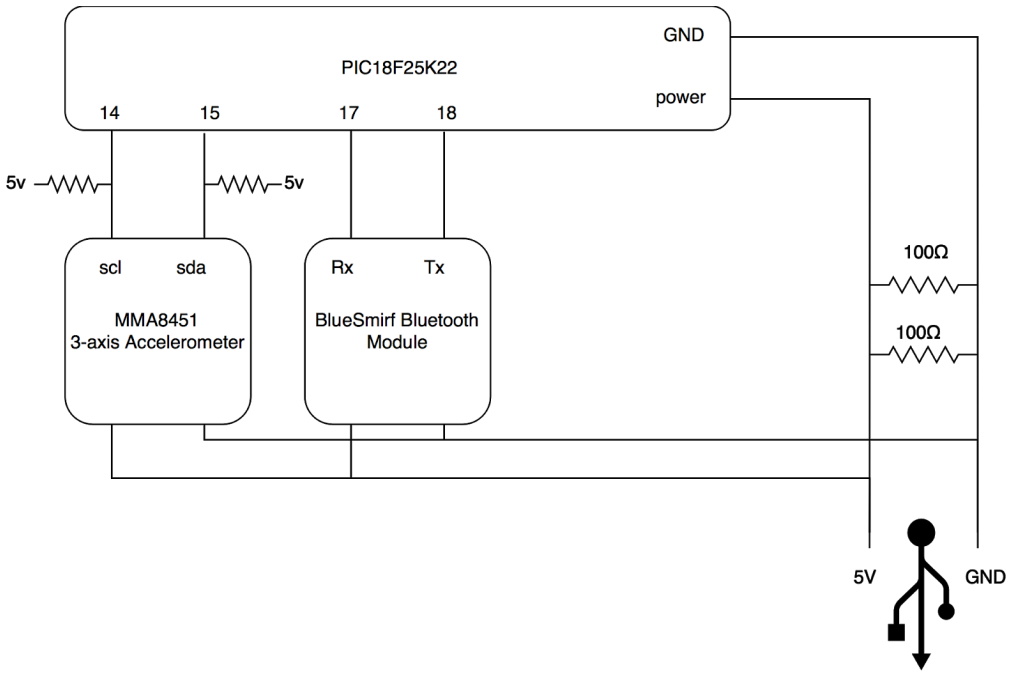

Figure 4: Hardware Layout of Bluetooth Remote

As the figure shows, the accelerometer is connected to the PIC via I2C, and the Bluetooth module is connected via the UART interface running at 115200 8N1. The two 100Ω resistors are placed across the power and ground lines to ensure that the device draws enough power from the USB cable to continually receive power. Otherwise, the port that the device is connected to will detect that no device is attached, and the power will be shut off.

PIC18F25K22

The PIC was the main controlling module in the bluetooth controller. It was programed using the PicKit 3 in-circuit debugger and programmer to get the software behaviour needed. For further details of its behaviour due to software, please refer to the software implementation section of this report.

MMA8451

The MMA8451 is a 3-axis accelerometer made by Adafruit. It is configurable into several different modes: +-2g, +-4g and +- 8g. Setting the mode indicates over what range the the accelerometer will measure acceleration (where g is gravitational acceleration). No matter which mode the chip is set to, the data will always be 14-bits in length. This means that the larger the measurement range, the smaller the measurement increments are. For this project, the MMA8451 was configured using software to be in +-4g mode. This was the ideal balance between accurate measurement and a large enough dynamic range. A picture of the sensor and the pinouts can be found in the figure below. For further information about the sensor, please reference the spec sheet located at: https://cdn-shop.adafruit.com/datasheets/MMA8451Q-1.pdf

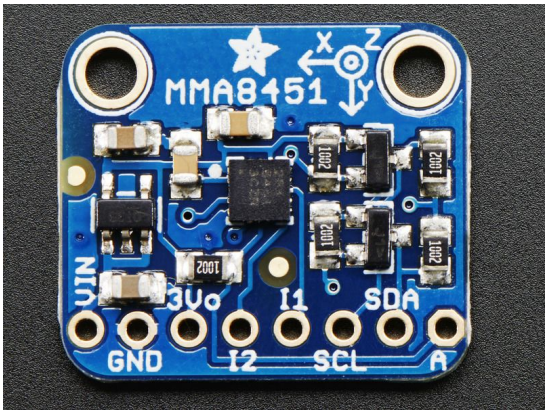

Figure 5: MMA8451 Accelerometer made by Adafruit

BlueSmirf Bluetooth Module

The BlueSmirf Bluetooth module is a module that converts the UART connection to a wireless bluetooth connection. For this project, the BlueSmirf was configured to slave mode and set to 1152008N1 for both the bluetooth connection and the UART connection. The figure below shows the BlueSmirf and its pinouts. For more information about this particular module, please refer to its data sheet located at: https://cdn.sparkfun.com/datasheets/Wireless/Bluetooth/Bluetooth-RN-42-DS.pdf

Figure 6: BlueSmirf Bluetooth Module Made by Sparkfun

Drive Control

The drive control hardware mainly consists of the L298N dual H-bridge and the 2 DC motors that are connected to it. The overall hardware layout is found in the figure below.

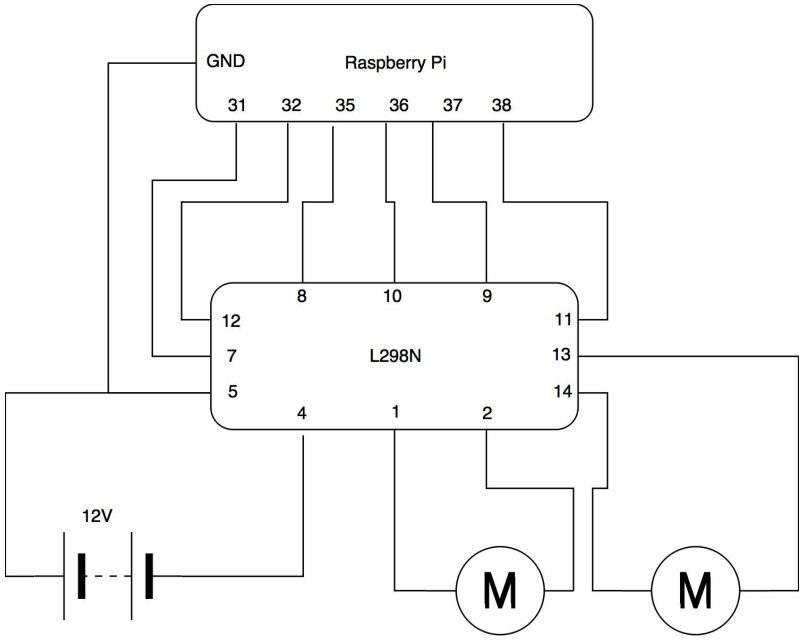

Figure 7: Hardware Layout for the drive Interface

As the diagram shows, the two DC motors are connected to the L298N H-bridge which is powered by the 12V battery pack. The raspberry pi GPIOs control the 4 select pins which control the rotation direction of the motors, and to the 2 enable pins which enable the motors to turn. By attaching PWMs to these enable pins, the motors are able to spin at variable speeds. For reference. The L298N module that was used in the project along with its pin numbers are shown in the figure below. The 12V DC battery pack consisted of 4 AA batteries in series.

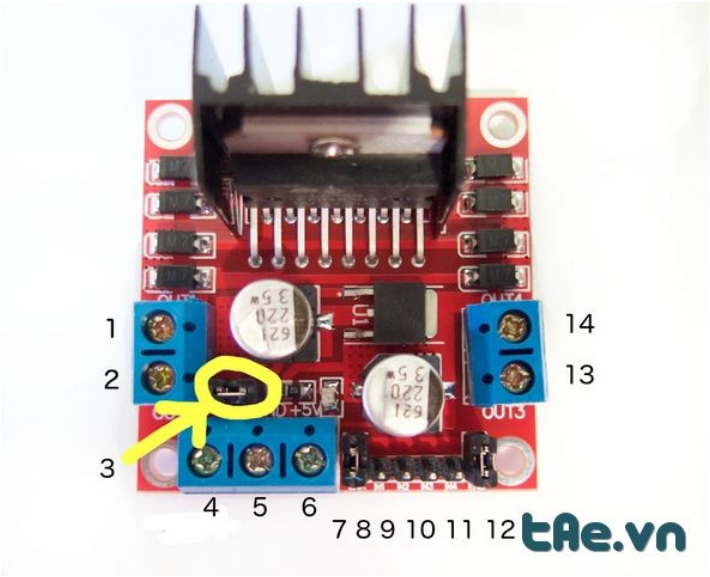

Figure 8: L298N H-bridge module

Chassis and DC motors

The chassis and DC motors used in this project came in a kit which utilized the 2 DC motor powered wheels to drive the vehicle, with a third smaller wheel used to balance the vehicle itself. The DC motors that were used had gearboxes built in to increase the torque of the motors. The chassis and motors uses are shown in the figure below.

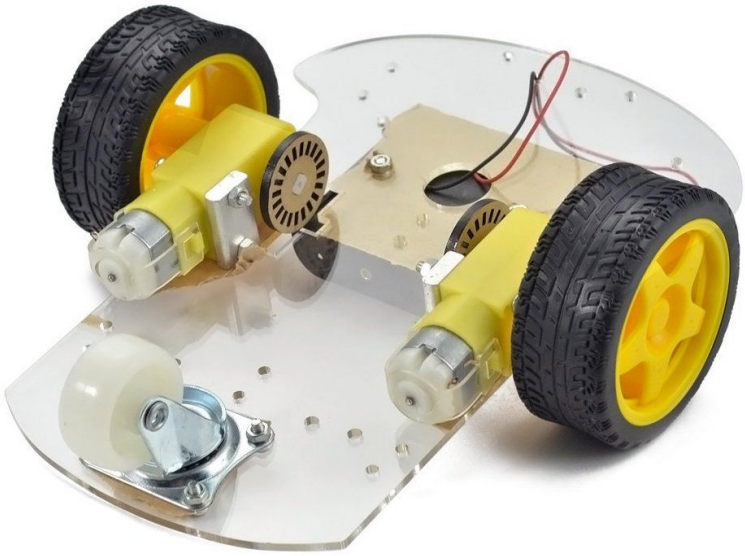

Figure 9: Chassis and DC motors used for the vehicle.

Webpage

Sensors

The SGP30 and SI7021 sensors onboard the vehicle were connected to the two I2C Pins on the Raspberry Pi, as well as to the 5V power and GND pins. The SCL and SDA lines were each connected to power via a 4.7kΩ pull-up resistor. This allowed communication with the two I2C sensors onboard the vehicle with minimum GPIO bank pin usage.

Servo-Control for the Camera

The pan tilt camera has two ribbon cables; the bottom one is for the “pan” servo and the top one is for the “tilt” servo. Each cable has three wires. The red wire for both the “pan” and “tilt” servos connect to the power supply. The brown wire for both the “pan” and “tilt” servos connect to the common ground. For the “pan” servo, the orange wire (control signal) is connected to GPIO 27 on the Raspberry Pi while for the “tilt” servo, the orange wire (control signal) is connected to GPIO 17 on the Raspberry Pi. The tilt-pan servo module that was used can be found in the figure below.

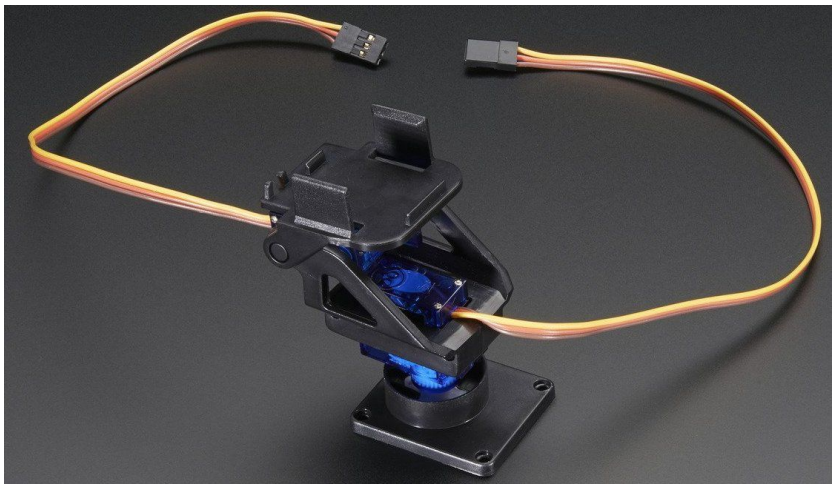

Figure 10: Pan-Tilt servo arm unit used in the project

Power

The Vehicle and the remote had three different power sources that are listed below.

- The remote was powered by a 5V 2A USB-based power bank
- The Raspberry Pi was powered by a 5V 2A USB-based power bank
- The DC motors were powered by 4 AA batteries in series.

DESIGN TESTING

Test Plan

The testing of the RC car was done in several stages. The first step was to see if the wheels on the car would turn and respond to the motion control device (established via bluetooth). This would mean that the car could turn right, left, go forward, or go backward. Additionally, the user should be able to control the speed of the car depending on the degree of tilt that is applied to the motion control device.

Aside from the movement of the RC car, it was also necessary to test whether the video page on the designed web page was clear and not suffering from any major lag. Furthermore, there was the added option of controlling the movement of the camera itself (through a servo). This involved testing whether the camera could look up, down, to the right, or to the left depending upon what option the user chooses on the web page. Lastly, the most important feature to be tested on the web page was whether it would display the correct temperature value, humidity, and CO2 and TVOC levels from the sensors placed upon the RC car. Any dangerous levels of air pollutants detected should result in a warning being displayed.

Test Specification

The system was tested for a wide array of functions. The first was testing to see if the tilt-control module would break in any case causing the vehicle to stop responding. The second was testing the video stream for lag (and clarity) by making the vehicle move quickly or change direction abruptly. The camera position was tested as well to see if it would pan and tilt correctly when the angle for either was incremented or decremented. The edges case for this stage involved to see how the system would respond if the user tried to pick an angle outside the range of 30

degrees to 150 degrees. Lastly, the air quality and temperature sensors were tested to see if they would change values (and display the appropriate warning) when the temperature was raised or isopropyl alcohol (for CO2 and TVOC) was introduced to an area near the vehicle.

Test Cases

Test Case ID	Test Description	Test Steps	Expected Results	Pass/Fail
0001	Testing if vehicle can move forwards and turn right	1. Tilt the motion control device forward and to the right.	Vehicle should move forward and turn right.	Pass
0002	Testing if vehicle can move forwards and turn left	1. Tilt the motion control device forward and to the left	Vehicle should move forward and turn left.	Pass
0003	Test video feed	1. Go to http://<RaspberryPI_IPAdress>:5000. 2. Check to see if video is streaming	Video should be streaming on the web page	Pass
0004	Testing if vehicle can move backwards	1. Tilt motion control device upwards/towards your body.	Vehicle should move backwards	Pass
0005	Testing if vehicle can move backwards and to the right.	1. Tilt motion control device upwards/towards your body and to the right.	Vehicle should move backwards and turn to the right.	Pass
0006	Testing if vehicle can move backwards and to the left.	1. Tilt motion control device upwards/towards your body and to the left.	Vehicle should move backwards and turn to the left.	Pass
0007	Test if camera	1. 1. Go to	Camera should	Pass

	(with attached servo) can rotate up	http://<RaspberryPI_IPAdress>:5000. 2. Click on “+” in the TILT angle row.	rotate upwards	
0008	Test if camera (with attached servo) can rotate down	1. 1. Go to http://<RaspberryPI_IPAdress>:5000. 2. Click on “-” in the TILT angle row.	Camera should rotate downwards	Pass
0009	Test if camera (with attached servo) can rotate to the right.	1. 1. Go to http://<RaspberryPI_IPAdress>:5000. 2. Press “+” in the PAN angle row.	Camera should rotate to the right.	Pass
0010	Test if camera (with attached servo) can rotate to the left.	1. 1. Go to http://<RaspberryPI_IPAdress>:5000. 2. Press “-” in the PAN angle row.	Camera should rotate to the left.	Pass
0011	Test if correct CO2 levels and warning message are displayed on webpage.	1. 1. Go to http://<RaspberryPI_IPAdress>:5000. 2. Use Isopropyl alcohol near the sensors.	Web Page should display the max CO2 concentration of 60,000 ppm with a level 4 (red) warning level.	Pass
0012	Test if TVOC levels and warning messages are displayed on the webpage.	1. 1. Go to http://<RaspberryPI_IPAdress>:5000. 2. Use Isopropyl alcohol near the	Web page should display display the max TVOC concentration of 60000 ppb and a	Pass

		sensors.	level 2 (red) warning level.	

PRESENTATION AND RESULTS

Our vehicle functioned as desired. All our implemented features were working correctly and displayed no problems. In user control, the vehicle changed direction correctly depending on how the motion control device was tilted. The web page was also displayed the up-to-date video-feed from the camera on the vehicle, allowing the user to see where he/she was going. We were even able implement the pan-tilt option for the camera as an addition to our initial proposed prototype. The user could choose to change the position of the camera through either the PAN or TILT option, in which case the angle of the camera (depending on the option chosen) would increment or decrement by 10 degrees. In both cases, the camera would change position in the correct manner.

Aside from these features, the data sensors were also working properly. When the system detected the user's breath or a substance like isopropyl alcohol, the values for temperature, humidity, CO2, and TVOC would change accordingly and the correct warning level was shown. All these results are shown in the pictures below.

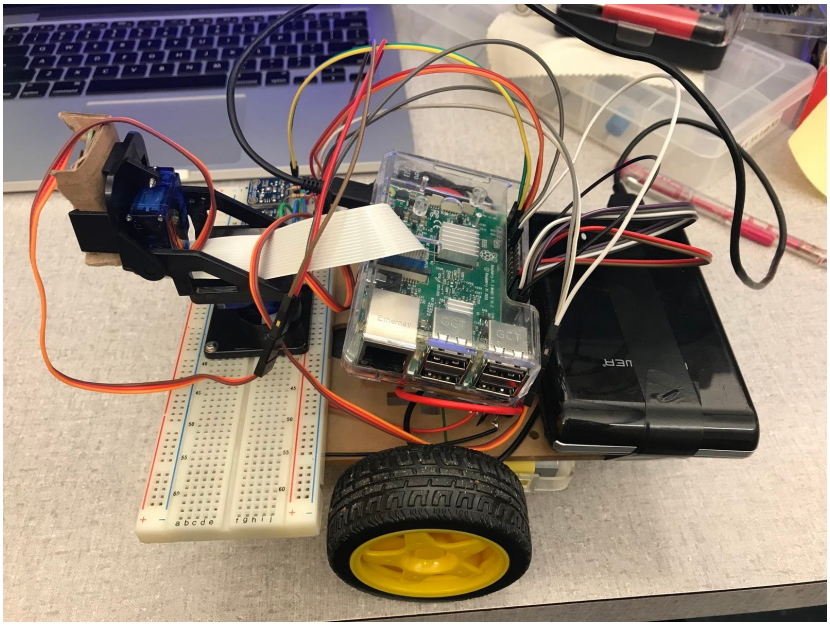

Figure 11: The overall prototype for the hazardous air vehicle.

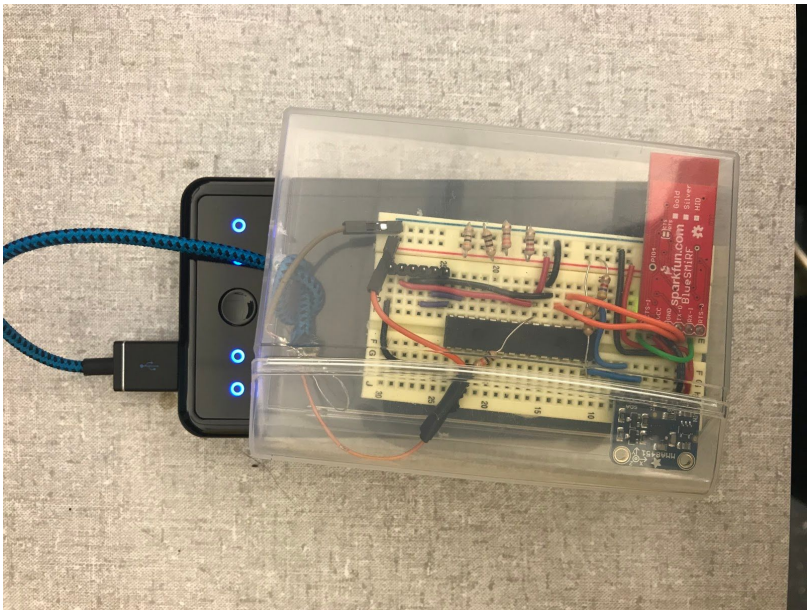

Figure 12: The tilt-based motion control device used to control the vehicle.

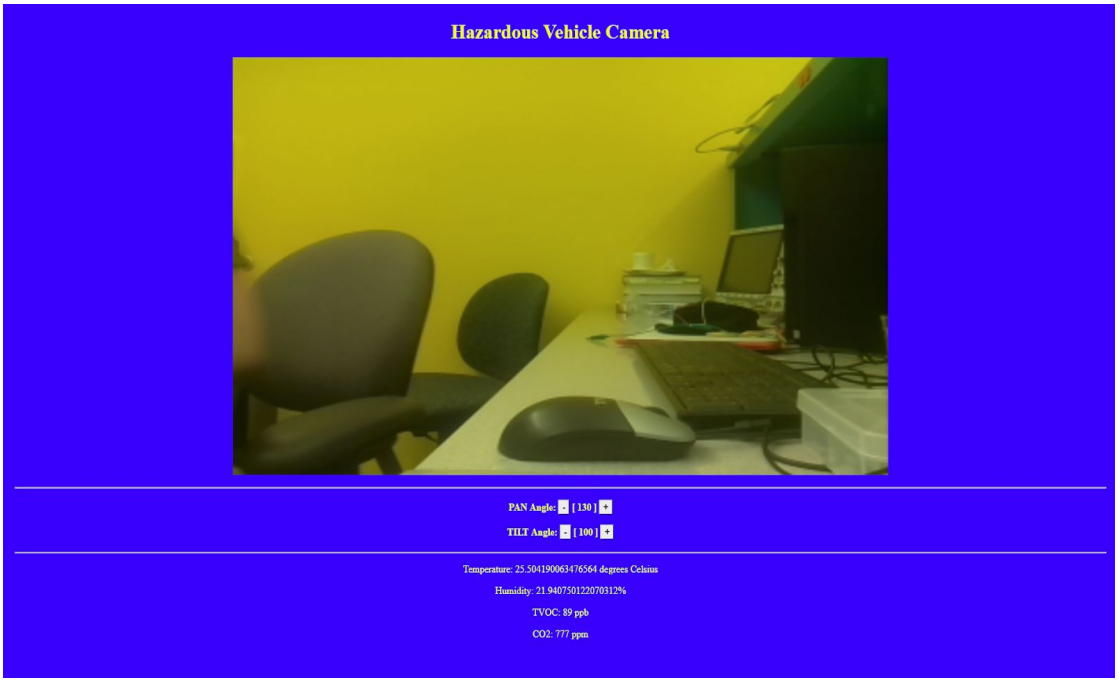

Figure 13: The complete webpage showing the video feed from the camera attached to the vehicle, the option to control the position of the camera through “pan” and “tilt”, and the data from the air quality and temperature sensors.

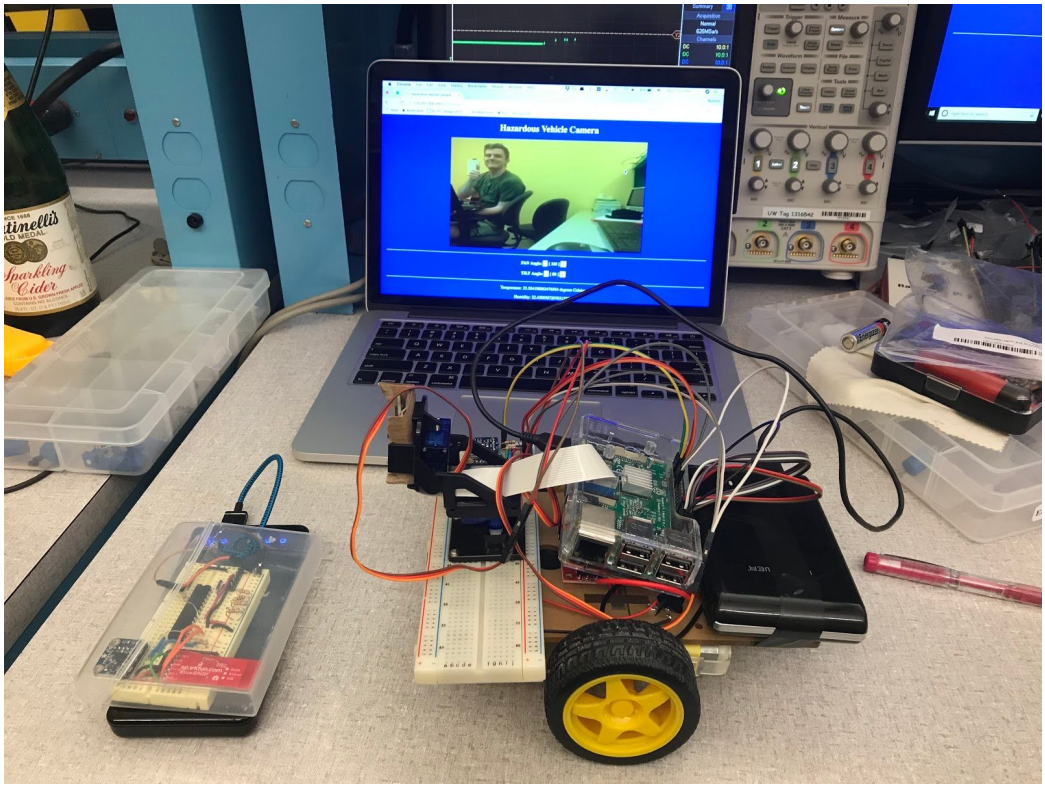

Figure 14: The video feed from the camera (pointed to the left) on the webpage.

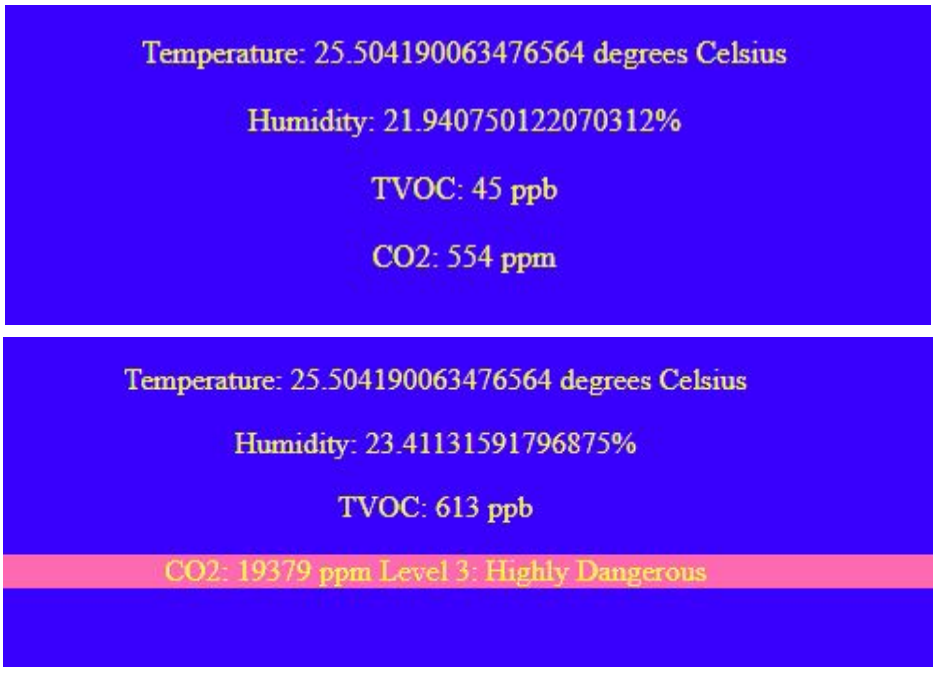

Figure 15: Shown is the before (top) and after (bottom) of the sensor data displayed on the webpage. This is in response to the user blowing on the sensors attached to the vehicle.

With the extensive tests that were done to allow the vehicle to remotely be controlled with the

remote via a video feed, all of the test cases were adequately tested for and satisfied. The vehicle was able to be controlled without visual contact by using the camera feed and the remote, and the sensor values were responsive and accurate.

ANALYSIS OF ERRORS/PROBLEMS AND FUTURE IMPROVEMENTS

Errors/Problems

There were a few small problems and roadblocks that were encountered in the course of this project. All of the issues were eventually resolved before the completion of this project. The errors that were encountered, along with the solutions, are found in the table below.

Table 8: Problems and Solutions in Project

Problem/Error	Solution
Unable to read data from the MMA8451 with the PIC	There were several issues involved in this. The first was that the MCC generated I2C library was not generating the proper sequence of I2C signals to read from the sensor. Once the library was rewritten, data was being read, but in a jumbled mess that made no sense. This was an error due to some typ casting in the libraries which was corrected.
Unable to connect the Raspberry Pi to the BlueSmirf module	The issue was that the Raspberry Pi would seemingly not connect to the BlueSmirf module with an error stating “no valid interfaces on device”, as if the Pi could not recognize that there was a serial interface on the BlueSmirf. However once some configurations were adjusted, the Pi was able to successfully pair with the BlueSmirf module
Unintelligible data being sent from the remote to the vehicle	The issue was that the data was being formatted and sent properly from the remote, but the vehicle was not receiving the same values as those that were sent at the remote. The values would definitely change, but in seemingly unpredictable ways when compared to the data at the remote. The issue was that the unpack() command in python was using a different endianness for decoding the byte data than expected which was causing the bytes to be decoded out of order. To adjust for this, the least significant byte was sent first, followed by the most significant byte. This corrected the error.
Corrupt data causes Python Drive Script to crash	The commands were being properly interpreted, but about once every 10 seconds, the program would crash with an error alerting the program that the unpack() command was unable to unpack the data due to an unexpected number of bytes. This error was attributed to the data being corrupted and the ‘/r/n’ characters of one sequence not being

	recognized, resulting in a readline() of 16 bytes which could not be unpacked with the designated format. In order to ignore this error, a try/except block was added to the code to ignore the error. This prevented the program from crashing when an error was encountered and allowed it to continue on.
Python Script does not release resources	The error was that whenever the python drive script was "killed" by the user, the GPIOs were not released as they should have been. This caused future executions of the same script to throw errors that the desired GPIOs were already in use. This was solved by creating a signal listener which listens for SIGTERM and adequately releases the appropriate resource when the script is killed unexpectedly.
Sensor Code is unable to detect SGP30 sensor	The error was that after initialization, the SGP 30 sensor was not able to be detected, read or written to using the python script. This was an error that was caused by a erroneous write to a register in the SGP30 which caused the sensor to become unresponsive. Once this command was taken away, the sensor functioned properly.
Controller is unable to be powered by USB	The error was that the bluetooth remote would be powered for about 15 seconds from the power bank before shutting off. The error was caused by the fact that the remote did not draw enough current from the power bank for it to recognize that a device was connected. To correct this, two 100Ω resistors were added in parallel with the devices to draw enough current from the power bank to get continuous power.

Future Improvements

Overall, we faced no real hurdles and serious errors during this project. There were several minor aspects however which could be refined for a future project. These improvements are listed in the table below.

Table 9: Possible Improvements to the Project in the future

Improvement	Description
Better web framework for the videostream	We used Flask for this project, which worked pretty well overall. However the video stream still suffered due to lag at times, such as when the camera position was changed. But this was not a noticeable issues and only occurred rarely. Using a better web framework could potentially decrease the lag times in the video and increase performance.

Use of IR Camera and/or motion detection	Since our vehicle was designed to operate in hazardous environments, it would be appropriate to add a IR camera or even motion detection in case the vehicle were to enter an area with extremely low visibility (very common in industrial cities with smog).
Change Drive Train to Caterpillar Tracks	Our current vehicle has three wheels which would have a very difficult time on rough terrain. By replacing these with caterpillar tracks, the vehicle would be able to navigate over much rougher terrain to accomplish its mission.
Make Vehicle Waterproof	The current prototype is very exposed to the elements and could not deal with any kind of moisture, wind, etc. To improve upon the design, the vehicle could be made to be full weatherproof to meet the needs of the user and operate in harsh environments.
Long range communication	The vehicle currently operates using WiFi 802.11N and Bluetooth 4.1 whose ranges are limited. Newer long-range protocols could be used to replace these protocols in future iterations to increase the range of the vehicle to operate further away.
Specialized Sensors	Currently the sensors only detect TVOC and CO2 which are rough estimates of the real breathing hazard in air. To get a more accurate idea of the harm to human health, more accurate sensors could be placed on the vehicle to measure elements like carbon monoxide and dust particles.
Advanced Gesture Based Control	Currently the vehicle is controlled by a tilt-based sensor on the remote, but this could be replaced with a more advanced gesture driven control scheme such as the myo to add more functionality and move the vehicle more seamlessly.

ANALYSIS OF WHY THE PROJECT MIGHT NOT HAVE WORKED

All components functioning properly.

SUMMARY

This report walks through the implementation of the hazardous air vehicle, going from detailing the system specification to covering hardware/software implementation and then to finally showcasing the results. The objective of the system was to provide the user a highly-reliable exploration vehicle which would provide accurate information on the air quality, temperature, and humidity of the surrounding environment (with the accompanying warning levels) while also producing a live video feed in which the user could control the position of an attached camera . All of this information was displayed on a webpage for the user's convenience.

CONCLUSION

This project was the culmination of everything we have learned so far. Aside from the PIC microcontroller, the use of a Raspberry PI, air quality and temperature sensors, an H-Bridge, and an accelerator were all required (along with other materials). The most interesting aspect of the final project was building an embedded system of our choice from scratch. We were able to meet all the goals and functionality initially desired for the vehicle while also having the opportunity to implement additional features such as the servo-arm to control the position of the camera. Furthermore, new skills gained were gained from learning how to set-up a webpage which could continuously display live information and video from another system. Overall, we are very happy with how our project turned out!

CITATIONS

none

APPENDIX A: Requirements Specification for a Hazardous Air Vehicle

Requirements Specification

System Description

This specification describes a prototype implementation of a remotely controlled, short-range, hazardous air vehicle. The prototype will consist of a vehicle with air quality sensors, temperature sensors and an on-board camera that can provide video and air quality feedback to the user while being controlled via a tilt-based control module. Additionally, the user can control the position of the camera through the webpage. The user will be able to use this vehicle to navigate into environments where the air quality is potentially hazardous, and monitor the surroundings through video and sensor readings to determine if it is a safe environment to enter.

Specification of External Environment

The prototype system is a proof of concept for control, communication, and environmental monitoring. As a result, the system must operate in a controlled laboratory environment with minimal outside temperature or other conditions negatively affecting the operation.

Temperature: 15 - 25 Celsius (Room Temperature), no humidity/moisture

System Input and Output Specification

System Inputs

Drive Control (User Input) - The system will support user control of the hazardous air vehicle via a wireless hardware controller. This hardware controller will implement tilt-based driving to allow the user a seamless driving experience using one hand.

Camera Position (User Input) - The system will support user control of the position of the camera. The user can choose to pan the camera position to the right by 10% or to the left by 10% or tilt the camera position to the top by 10% or to the bottom by 10%.

System Outputs

Video Feed (User Output) - The system will support feedback to the user about the system and surrounding environment via a website. This website will have a live video feed from cameras attached atop the hazardous air car vehicle so that the user can survey the surrounding environment visually.

Amount of Air Pollutants: The system will also provide the user with data on the amount of air pollutants outside via the website. CO, CO2 and VOC (volatile organic compounds) that are harmful to human breathing will be the three main pollutants being measured. The user will be provided with live data concerning the air quality, and will be warned immediately if pollutant levels exceeding healthy parameters.

Temperature and Humidity: In addition to the pollutant levels and the video feed, the system will display the current temperature and relative humidity of the surrounding environment for the user to see. This information will be displayed on the website.

Camera Position: The webpage will also display the current position (in degrees) the camera is at for both pan and tilt.

User Interface

The user interface in this project will consist of two parts:

- Control of the hazardous vehicle via the hardware remote.
- Monitoring of the surrounding environment via the Website.

The Website is reserved for feedback to the user and will not have any control over the movement of the car but will have the option of changing the camera position. The movement control for the car will be solely based on the commands provided by the hardware remote.

The hardware remote will provide the following functionality for driving the vehicle:

- Drive straight (forward/back)
- Turn while driving forward/back (left/right)

System Functional Specification

The system is intended to provide a working prototype of a hazardous air vehicle. There will be three main components to it.

1. The vehicle component. The prototype vehicle will have an on-board camera as well as a CO/CO2/VOC sensor to detect air pollution levels and a temperature/humidity sensor for environmental information.
2. The wireless tilt-based control module (called the wand), which allows the user to drive the vehicle. This module will send data to the vehicle component in order to accurately control its movement.
3. The website which will receive a live video feed from the car, as well as data from the sensors regarding the amount of air pollutants outside. If the RC car is in an environment where the air pollution poses a health threat, the website will alert the user with various warnings, telling him/her to leave that area as soon as possible. The website will also display information of the camera position and allow the user to change it.

Operating Specifications

As a prototype the system will operate in a controlled laboratory environment.

Temperature: 15 - 25 Celsius
Humidity: None
Power: 5V DC Power on vehicle control, 6V DC power supply on motors
Power (Wand): 3.3V DC power supply

Reliability and Safety Specification

The system will be designed to be as reliable and efficient as possible in terms of accurately reporting the state of the environment around the user. It will issue warnings to the user if the air pollution in a given place is too high. Aside from this, the system will also be reliable in moving where the user wants at the desired speed.

Since the system will operate in the U.S. , it will comply with the following specifications and regulations:

- 47 CFR 15: Code of federal regulations for unlicensed transmissions
- 802.11 b/g/n Wireless LAN
- Bluetooth 4.1
- 15 U.S.C. 1278a

APPENDIX B: UML use cases for Hazardous Air Vehicle

Bluetooth Remote

The UML use case diagram for the Bluetooth remote of the system is show below.. Additionally, each use case is textually described following the figure.

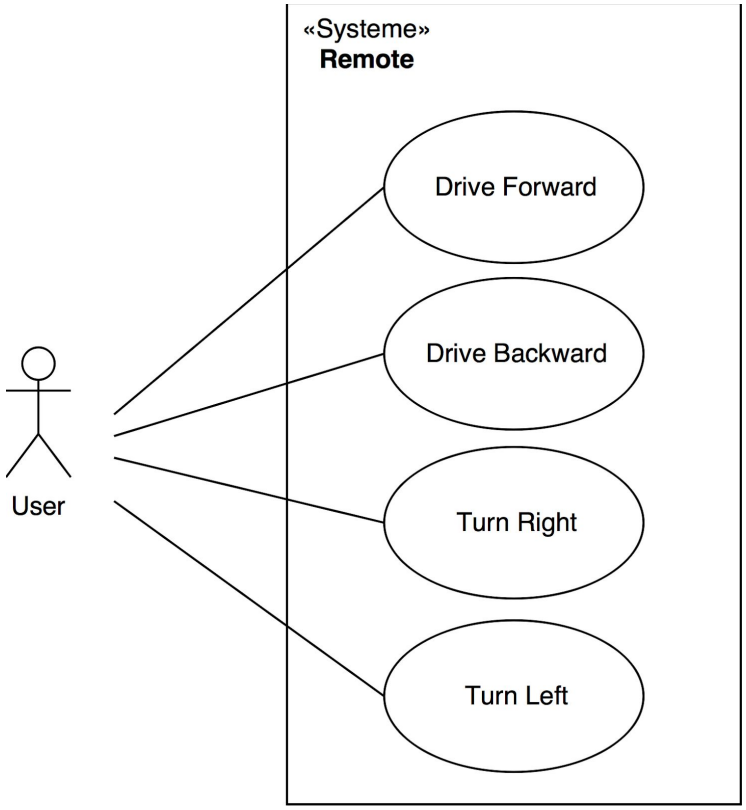

Figure 1: UML use cases for the bluetooth remote

Drive Forward
This use case allows the user to drive the vehicle forward at varying speeds depending on the degree of tilt applied to the remote.
Exceptions: When the remote is disconnected from the vehicle Bluetooth.

Drive Backward
This use case allows the user to drive the vehicle backwards at varying speeds depending on the degree of tilt applied to the remote.
Exceptions: When the remote is disconnected from the vehicle Bluetooth.

Turn Right
This use case allows the user to turn right while driving the vehicle forward (at varying speeds) depending on the degree of tilt applied to the remote.
Exceptions: When the remote is disconnected from the vehicle Bluetooth.

Turn Left

This use case allows the user to turn right while driving the vehicle forward (at varying speeds) depending on the degree of tilt applied to the remote.
Exceptions: When the remote is disconnected from the vehicle Bluetooth.

Vehicle
The UML use case diagram for the vehicle portions of the system is shown in the figure below. Additionally, each use case is textually described following the figure.

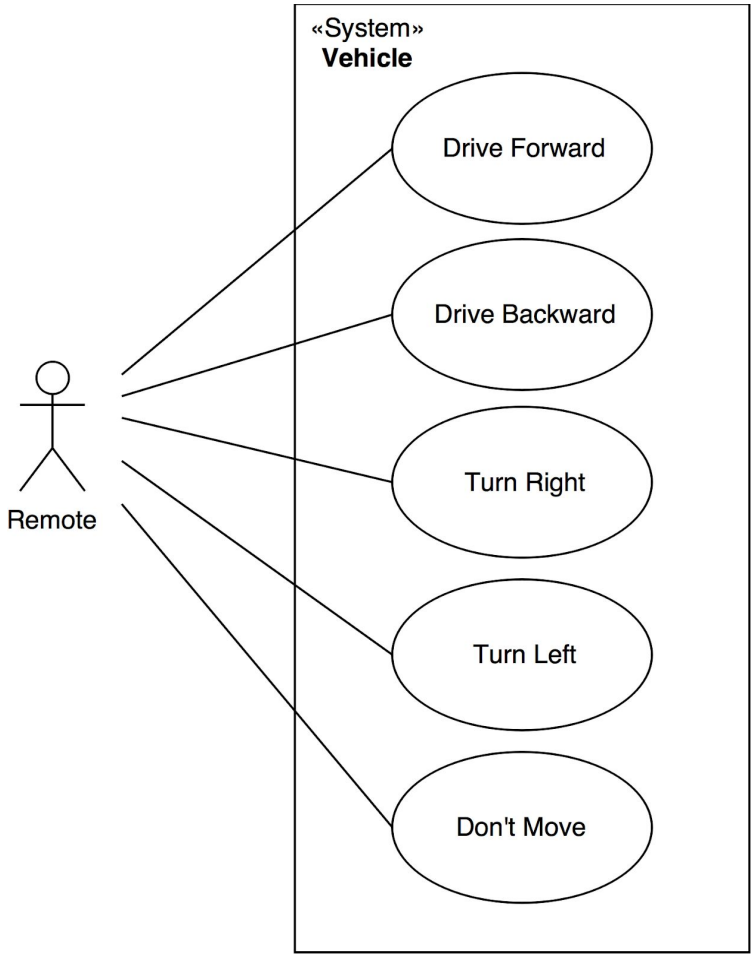

Figure 2: UML use cases for Vehicle control channel

Drive Forward
This use case allows the remote to set the motor speed and direction to drive the vehicle in the forward direction.
Exceptions: When the remote is disconnected from the vehicle Bluetooth. H-bridge module or motors have hardware level control errors.

Drive Backward
This use case allows the remote to set the motor speed and direction to drive the vehicle in the backwards direction.
Exceptions: When the remote is disconnected from the vehicle Bluetooth. H-bridge module or motors have hardware level control errors.

Turn Right
This use case allows the remote to set the motor speed and direction to allow the vehicle to turn right while driving forward.
Exceptions: When the remote is disconnected from the vehicle Bluetooth. H-bridge module or motors have hardware level control errors.

Turn Left
This use case allows the remote to set the motor speed and direction to allow the vehicle to turn leftt while driving forward.
Exceptions: When the remote is disconnected from the vehicle Bluetooth. H-bridge module or motors have hardware level control errors.

Don't Move
This use case allows the remote to turn the motors off so that the vehicle is completely stationary.
Exceptions: When the remote is disconnected from the vehicle Bluetooth. H-bridge module or motors have hardware level control errors.

Website - User Feedback
The UML use case diagram for the website that is used for user feedback is shown below. Additionally, each use case is textually described following the figure.

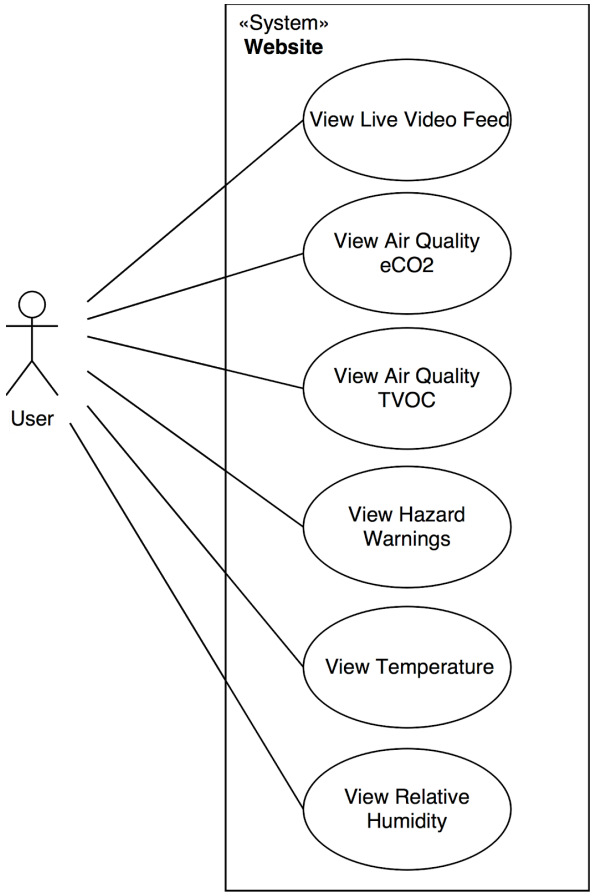

Figure 3: UML use cases for Website

View Live Video Feed
Allows the user to view the live video feed from the vehicle.
Exceptions: App is disconnected from the vehicle wifi access point.

View Air Quality eCO2
Allows the user to view the air quality reading of effective CO2 from the vehicle in parts per million.
Exceptions: App is disconnected from the vehicle wifi access point.

View Air Quality TVOC
Allows the user to view the air quality reading of total volatile organic compound from the vehicle in parts per billion.
Exceptions: App is disconnected from the vehicle wifi access point.

View Hazard Warnings
Allows the user to be notified of hazardous air conditions based on the sensor readings and healthy contamination levels.
Exceptions: App is disconnected from the vehicle wifi access point.

View Temperature
Allows the user to view the temperature of the surrounding environment in C.
Exceptions: App is disconnected from the vehicle wifi access point.

View Relative Humidity
Allows the user to view the relative humidity in % of the surrounding environment.
Exceptions: App is disconnected from the vehicle wifi access point.

APPENDIX C: Software Functional Decomposition

Functional Decomposition

Bluetooth Remote

The following figure breaks down the software of the bluetooth remote into its main functional components. For further overall functionality of this component, please refer to the design specification.

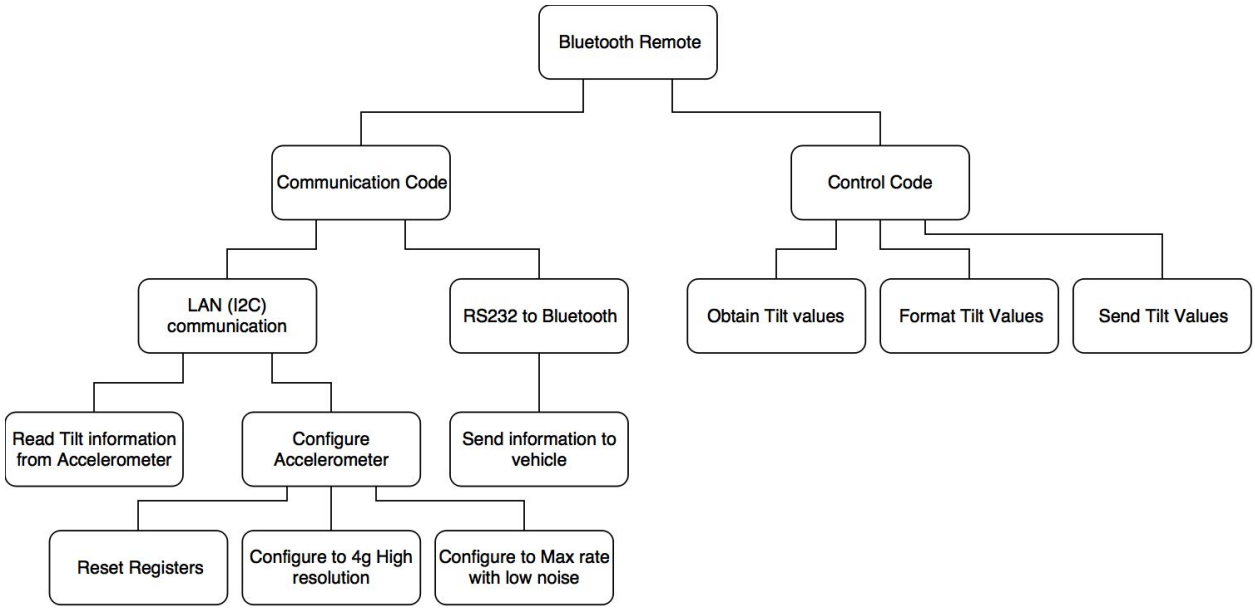

Figure 1: Bluetooth Remote Software Functional Decomposition

Vehicle

The following figure breaks down the software of the vehicle into its main functional components. For further overall functionality of this component, please refer to the design specification.

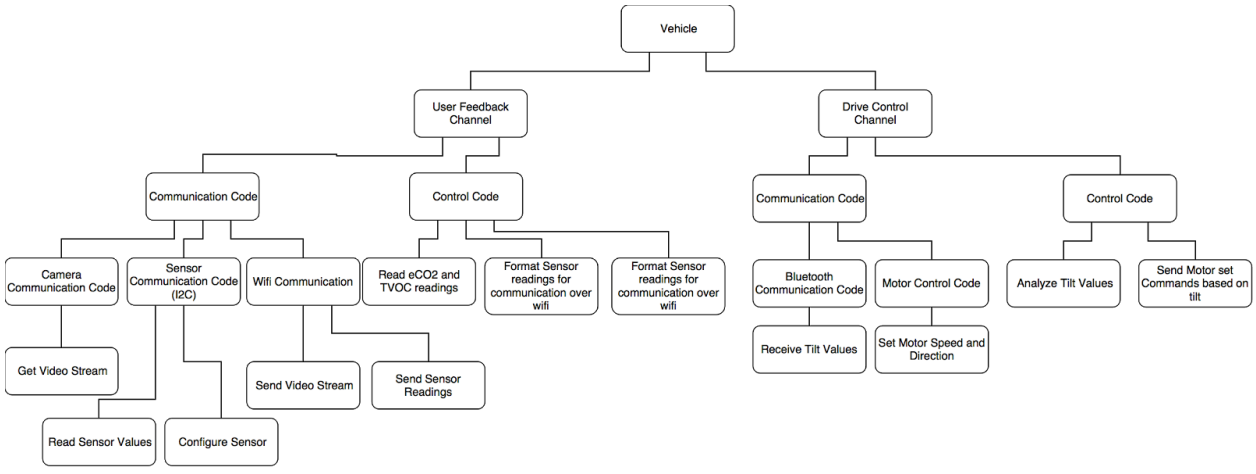

Figure 2: Vehicle Software Functional Decomposition

Website

The following figure breaks down the software of the website into its main functional components. For further overall functionality of this component, please refer to the design specification.

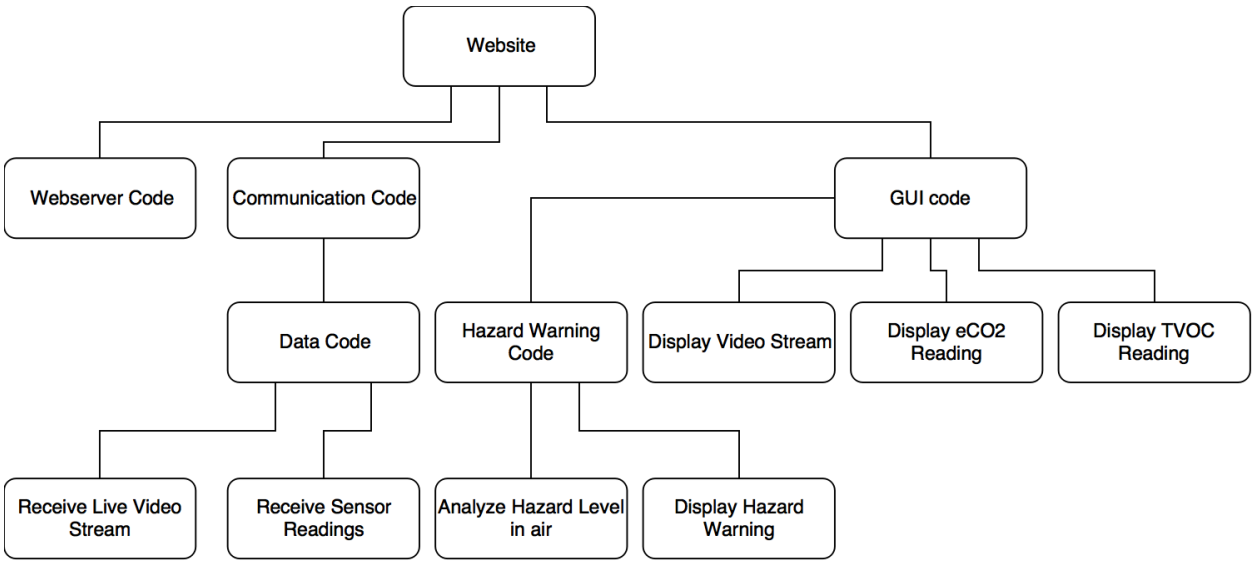

Figure 3: Website Functional Decomposition

APPENDIX D: Bill of Materials

Part	Number	Price
Raspberry Pi 3	1	$49.99
32 GB micro SD card	1	$11.99
PIC18F25K22	1	$4.80
4.7kΩ resistor	2	$0.20
100Ω resistor	4	$0.40
33uF capacitor	1	$0.30
Vehicle Chassis + motors	1	$12.59
L298N H-bridge	1	$6.89
Adarfruit MMA8451 Accelerometer	1	$8.95
Wiring	-	$2.00
Adafruit SGP30 air quality sensor	1	$19.95
Arducam 5mP	1	$13.49
AA Battery	4	$2.00
Adafruit SI7021 temperature Sensor	1	$6.95
Pan/Tilt Servo Arm	1	$16.95
BlueSmirf	1	$24.95
	Total:	$182.40

APPENDIX E: Failure Mode Analysis

FMEA Form

Process/Product Name: Hostile Environment RC Car

Prepared By: Sam Johnson, Emraj Sidhu

Responsible: Sam Johnson, Emraj Sidhu

FMEA Date (Orig.): 18-Feb (Rev.):

Process Step/Input	Potential Failure Mode	Potential Failure Effects	SEVERITY (1 - 10)	Potential Causes	OCCURRENCE (1 - 10)	Current Controls	DETECTION (1 - 10)	Action Recommended	Resp.	SEVERITY (1 - 10)	OCCURRENCE (1 - 10)	DETECTION (1 - 10)
What is the process step, change or feature under investigation ?	In what ways could the step, change or feature go wrong?	What is the impact on the customer if this failure is not prevented or corrected?		What causes the step, change or feature to go wrong? (how could it occur?)		What controls exist that either prevent or detect the failure?		What are the recommended actions for reducing the occurrence of the cause or improving detection?	Who is responsible for making sure the actions are completed?			
Loss of Power	System could fail or there could be a general power outage	RC Car will stop operating	9	Total system failure or a mass power outage	2	None	2	Have a second voltage supply nearby to use in case of failure.	Both team members.	9	2	2
Tilt control does not work and user cannot steer car	User will not be able to control the RC car as desired	RC car may not move or go in the incorrect direction (possibly crashing into something)	7	Faulty software which causes car to respond incorrectly to user command. Bluetooth connection between PIC and raspberry pi not	3	Backup app which allows user to control the RC car in case the first source does not work	8	Plenty of testing should be done in order to see car steers in correct direction.Bluetooth connection should be double checked.	Both team members.	7	3	8

				working properly				And if it is still faulty, use the app as backup				
Sensors collect incorrect data	Sensors could be connected incorrectly to the rest of the system or may otherwise be broken	User will not know the correct CO2 levels outside the vehicle	8	Sensors may not be connected properly. Also communication with the app may not work properly due to incorrect setup	6	Use of two sensors (possibly) with each one collecting data independently and relaying that information to the user. That way error checking can be done in comparing the data	4	As spoken previously, use of more more sensors will be recommended so user can compare and contrast data in order to see if one of them is faulty. Another recommend course of action would be to thoroughly test the sensor before final use	Both team members.	8	6	4
User gets no video stream	User will not have the capability of viewing the environment from the camera of the RC car	In area where this is smog, user will have difficulty navigating the surrondin g area without any proper	9	Bad wi-fi connection between phone and raspberry pi camera	6	In preventing failure, the recommended course of action will be for the user to maintain a close distance to the rc car	5	In case android app does not work, stream the video onto a web app or use the RaspiCam remote app in order to get	Both team members.	9	6	5

		visual						alernative method for video streaming				