Mars Mission Emulation Framework

Abdullah Hendy[#1] and Michael Granberry[#2]
[#]Electrical and Computer Engineering Department, California State University Northridge, USA
Faculty Advisor: Shahnam Mirzaei, Ph.D.
[1]Abdullah.Hendy.30@my.csun.edu, [2]Michael.Granberry.612@my.csun.edu

***Abstract*—The primary objective of this project is to build a simple yet comprehensive framework that emulates one of the National Aeronautics and Space Administration (NASA)'s missions on Mars. The addressed mission primarly uses the helicopter Ingenuity and the rover Perseverance to explore Mars. Similarly, the developed framework uses a drone equipped with a single-board computer as the aerial vehicle and a wheeled mobile robot mainly utilizing a bare-metal microcontroller and multiple sensors as a ground vehicle. The job of the aerial vehicle is to scan the environment and detect certain objects. The job of the ground vehicle is to receive commands from the aerial vehicle and execute tasks. The communication between the two platforms is wireless. An important objective of this project is to expose students and new graduates to an environment similar to a realistic high-end operation. This includes working with a complete system featuring concepts like aerial systems, machine learning, computer vision, sensor integration, robotics, wireless communication, and Linux-based operating systems.**

***Keywords*—robotics, drones, Pixhawk, artificial intelligence, single-board computers, Linux, Raspberry Pi, computer vision, Raspberry Pi camera, 3D printing, drone calibration, embedded systems, MSP432, Bluetooth, HC-05, Mars mission, wireless communication, Wi-Fi, LoRa, F', Edge TPU, Google Coral, UART, timers, pulse width modulation, sensors, analog to digital conversion.**

I. Introduction

This project aims to emulate one of the main operations of the helicopter Ingenuity and the ground vehicle Perseverance on Mars. That is, the Ingenuity scans the land to determine if an object or a spot is worth exploring by Perseverance. The aerial vehicle also checks if it is safe for Perseverance to take a certain path to reach the desired destination. The framework developed by this project achieves this scenario by dividing the system into two major blocks: ***Aerial Platform*** and ***Ground Platform***. Within each of the major blocks, there two sub-blocks are: ***Hardware*** and ***Software/Firmware***. The communication channel between the major blocks is wireless. In this implementation of the framework, the role of the ***Aerial Platform*** is to scan the test environment and send commands over Bluetooth to the ***Ground Platform*** based on the detected objects. The role of the ***Ground Platform*** is to execute tasks based on the different commands received.

The ***Aerial Platform*** is the Hawk's Work F450 drone platform that is based on a clone of the Holybro Pixhawk flight controller. The ***Hardware*** block consists of external modules used with the Pixhawk clone including a GPS module, safety switch, buzzers, radio control (RC) receiver and transmitter, and other modules.

The ***Hardware*** block also includes the Raspberry Pi 4 single-board computer [11], Raspberry Pi Camera, and the Google Coral tensor processing unit (TPU) [3]. The ***Software/Firmware*** block consists of calibrating and configuring the flight controller as well as other Raspberry Pi related pieces of code. First is the code used to train a custom TensorFlow Lite model for the operation environment. Seconds is the code used to allow the camera to detect objects using TensorFlow Lite and OpenCV along with other useful camera libraries. The final piece of code is to set up the Raspberry Pi Bluetooth core for wireless communication.

The ***Ground Platform*** is the Texas Instruments RSLK robot platform [2]. The ***Hardware*** block is mainly the TI MSP432P401R development kit along with the robot chassis, modules like the Pololu GP2Y0A21YK0F inferred analog distance sensor and a Bluetooth module. The ***Software/Firmware*** block is the firmware loaded onto the MSP432 microcontroller to allow the robot to collect data from the sensors, generate PWM signals to drive the motors, drive indicator LEDs, and set up the serial Bluetooth Module.

The communication between the two major blocks is done wirelessly through Bluetooth. This is achieved by using the HC-05 Bluetooth serial module interfacing with the MSP432 microcontroller on the ***Ground Platform*** side [6]. On the ***Aerial Platform*** side, the Bluetooth support on the Raspberry Pi is utilized [11].

The initial framework developed in the project adopts a simple approach when deploying, testing, and operating the system. This is to set the foundation for the main elements and concepts for the framework before moving to emulate more complex situations and environments to emulate NASA's operations on Mars. The main environment is a semi-empty room with the objects being three cones with different colors. The artificial intelligence (AI) model was trained to differentiate between three different colored cones. The drone is supposed to roam a small area scanning for these cones. Upon detecting each cone, a separate command is sent to the ground robot to perform a specific task.

Finally, this framework sets the foundation to expand on the platforms, test environment, and the overall reliability of the system. These upgrades could be divided into three

categories: design upgrades, hardware upgrades, and software upgrades. Design upgrades could include using better wireless communication protocols, adding more complex objects to the operation environments, and finding more efficient ways of controlling the aerial platform. Another design addition would be introducing frameworks like Robotics Operating System (ROS) and NASA's ***F'*** (fprime) as the main communication method of the two vehicles on the same network [14]. Software upgrades could include replacing bare-metal programming with an operating system and training more robust AI models. Hardware upgrades could include upgrading the existing platforms to more powerful ones and integrating more sensors.

II. Overview

A. Block Diagram

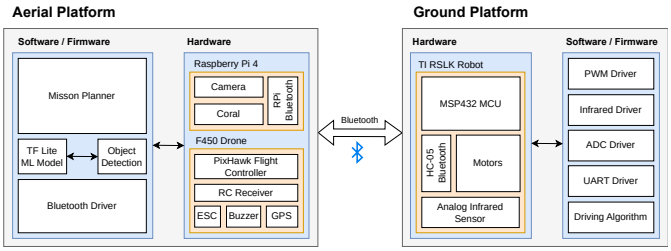


Fig. 1. A full block diagram showing the important parts of the system.

B. WorkFlow

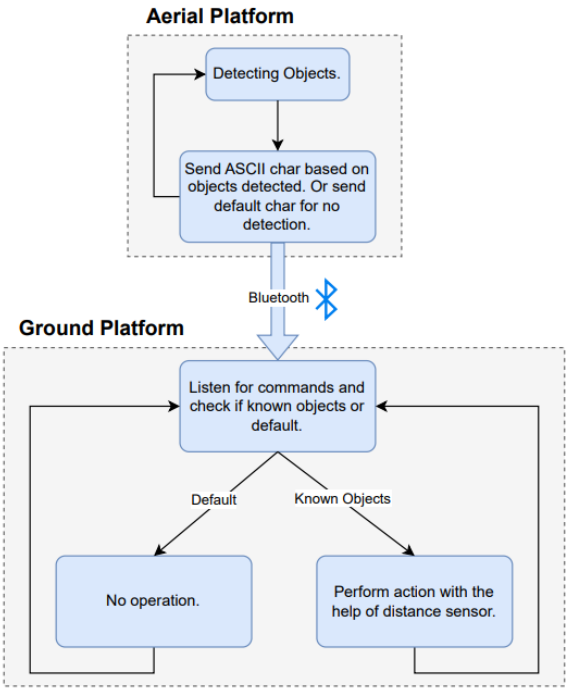


Fig. 2. A full workflow diagram showing how the system is operated as a whole.

III. Implementation

A. Aerial Platform

The platform used is the quadcopter F450 drone. This model was selected for many reasons that make it suitable for building this framework. The kit is widely adopted among developers with less experience in the aerial vehicles field, which makes finding documentation, tutorials, and debugging less of a complex task. The kit is also affordable, yet, it provides quality materials for the different parts. The F450 platform design allows for conveniently mounting a Raspberry Pi on top.

1) Hardware

The platform was brought up by first assembling the frame. The arms were mounted on the power distribution board (PDB) and were all screwed on the main base of the drone. The four motors were attached to the end of the arms where the propellers are to be mounted. However, the propellers were mounted last to ensure safety by testing all parts of the assembled drone first. The legs were installed to give the drone proper support and provide a lift, which allowed for mounting the Raspberry Pi Camera on the bottom of the PDB. The electronic speed controllers (ESC) were connected to the motors and the PDB.

The Raspberry Pi and the flight controller, the Pixhawk clone, were mounted on the main base with the Raspberry Pi below the controller. To properly mount the Raspberry Pi, a 3D part was printed to house it. This housing is made of two parts with shock absorbers installed in between to prevent vibrations from affecting the Raspberry Pi and the flight controller.

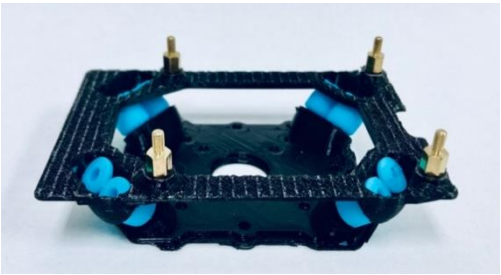


Fig. 3. The main 3D printed part mounting on the drone base, which holds the Raspberry Pi and everything above.

Fig. 4. The Raspberry Pi 4 mounted on the 3D printed part.

Another 3D plate was printed to mount the flight controller on top. Standoffs were used to create a proper space between the two platforms.

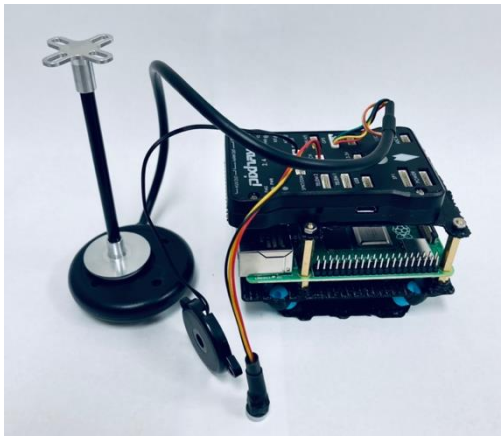


Fig. 5. The Raspberry Pi mounted on the base 3D printed part and the flight controller mounted on the other 3D printed part separated by standoffs.

After mounting the Raspberry Pi and the Pixhawk, some modules were connected to the flight controller. These modules include the ESC connectors, the RC receiver, a buzzer, a safety switch, and a GPS and compass module. For the camera, the Raspberry Pi Camera V2 was used instead of a newer module like Module 3 for compatibility issues that will be explained later. The camera was mounted underneath the drone base with the help of two 3D-printed sheets with vibration-dampening pads inserted in between. This was done to help get a good stable view of the test environment. The Holybro Power Module was connected to the battery connector and the flight controller, which helps regulate the power going to the flight controller as well as voltage and current sensing. Finally, an RC transmitter was set up and paired with the RC receiver. An RC transmitter is used as the controlling device for the drone in this implementation of the framework [5].

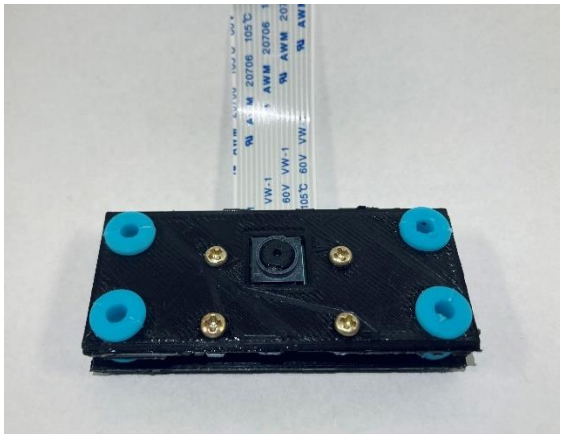
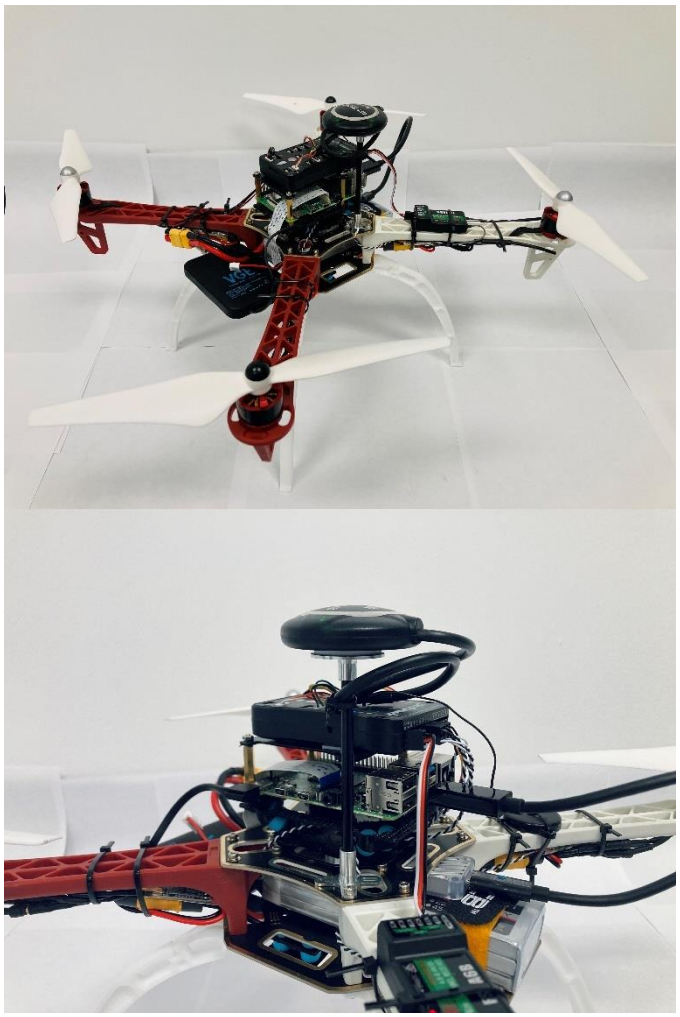


Fig. 6. The Raspberry Pi camera mounted between the 3D printed sheet with vibration dampening pads.

On the Raspberry Pi, the camera is connected through the Camera Serial Interface (CSI) using the Flexible Flat Cable (FFC). The Google Coral TPU is connected through one of the USB 3.0-supported ports on the Raspberry Pi. The following figure shows three views of the final drone build.



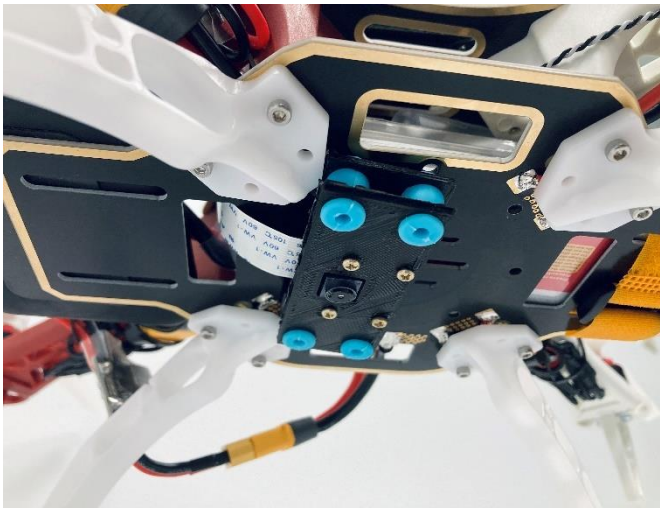


Fig. 7. The final build of the aerial vehicle. (Top):an overview of the platform. (Middle): a detailed view showing the different main components. (Bottom) the back of the platform where the camera resides.

The Raspberry Pi 4B with 8GB of RAM was used because of its capabilities, popularity, and ease of use that it provides. Although other single-board computers could be used, using the Raspberry Pi accelerates the process of development and debugging due to its large community support. It's easily booted from a micro-SD card. The Raspberry Pi has on-board Bluetooth support, which is compatible with the main communication used for the framework. Along with the Coral TPU, the Raspberry Pi is used as the brain of the aerial platform where environment scanning and object detection are executed.

2) Software/Firmware

The first step involving software was calibrating the Pixhawk clone flight controller and the modules attached to it. The calibration was done using the open-source software, Mission Planner, which provides a user-friendly interface for configuring, monitoring, controlling, and vehicle safety checking. First, the most recent version of the Pixhawk 1 firmware was flashed on the flight controller. This is the same Pixhawk version that the flight controller used clones. The next step was calibrating the accelerometer by adjusting the drone in different positions and following the instructions on Mission Planner. The compass was then calibrated by constantly moving the drone in all three-dimensional directions. The radio was calibrated by adjusting the switches and throttle control joysticks to their minimum and maximum positions on the RC transmitter. Flight modes were then selected including Altitude Hold, Stabilize, and Loiter. These modes were mapped to three different switch positions on the RC transmitter. Since the Pixhawk is a microcontroller-based flight controller, Mission Planner provides a list of registers that can be modified to set different functionalities. The registers were used to set critical battery voltage fail-safe values. The following figures show some calibration steps as well as register value modifications [1][5].

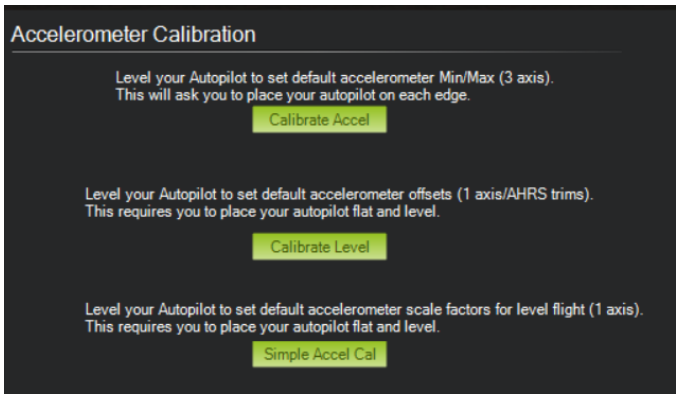


Fig. 8. Accelerometer calibration in Mission Planner.

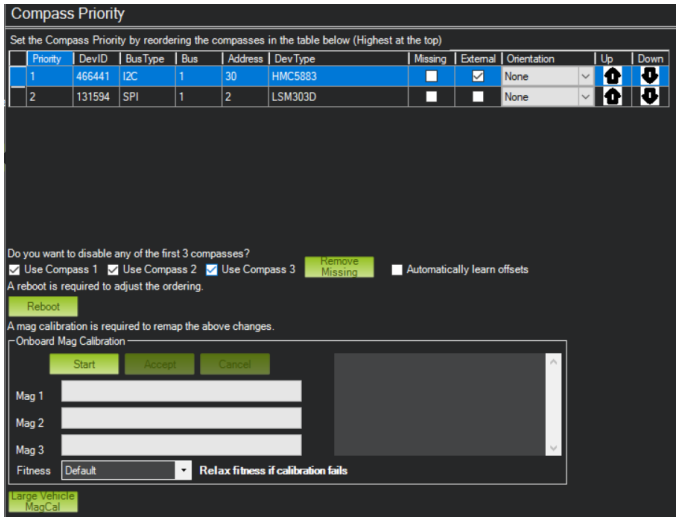


Fig. 9. Compass calibration in Mission Planner.

Fig. 10. Radio calibration in Mission Planner.

The battery was then connected, and the voltage/current sensor was calibrated by setting up the correct voltage divider

value. The motors were tested using the provided tool in Mission Planner to check if the motors rotate in the correct directions compatible with how the RC transmitter joysticks were configured. Finally, with the safety switch on, the drone was armed, and the voltage, GPS, compass, motors, and RC transmitter-receiver were checked for functionality. Since the flight controller used is a clone of the Pixhawk, firmware issues lead to unstable readings from the accelerometer and the gyroscope, which in turn, causes pre-arm safety issues to get fired. The workaround for this issue was to disable pre-arm safety checks while taking extra safety measures when flying the drone. This was done by writing a **0x0** to the ***ARMING_CHECK*** register.

Fig. 11. Main view from Mission Planner showing the flight controller statuses and the GPS location.

The second part of the software development was training an AI model that understands the test objects, which are to be detected by the drone. For this implementation of the framework, the objects are red, yellow, and green cones as well as the TI RSLK robot. The first step of the training process was to take multiple pictures of the desired objects in different situations, for example, different angles, lighting, and distances. About 300 pictures were taken in total of three different cones: red, yellow, and green, in addition to the TI RSLK robot. These pictures were prepared for training using an open-source labeling tool, which generates an XML file containing labeling details. The preparation process includes removing unwanted pictures and drawing boxes around the desired objects [2].

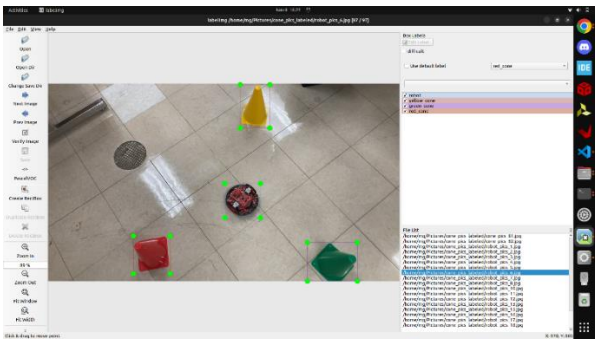


Fig. 12. Labeling process and data preparation before training the model using LabelImg open-source tool.

The next step was to train the model. The training process was done in a Google Collab environment, which provides renting services for high-quality processing units. TensorFlow 2 template codes provided by Google were used to train the prepared images. From the TensorFlow 2 model pool, the ***SSD MobileNet V2 FPNLite 320x320*** model was used for training. This is a lightweight model that is suitable for embedded applications and object detection applications running on less powerful computers such as the Raspberry Pi [2][12]. The model was converted into a TensorFlow Lite model and was then quantized by converting its weights from 32-bit floating numbers to 8-bit integer values. Although this results in a partial loss of accuracy, the quantization was performed to make the model compatible with the Google Coral TPU, which only accepts models with 8-bit integer weights. The following figure shows the TensorBoard monitoring tool, which was used to keep track of the training process. The model was trained until the total loss got to about 0.06.

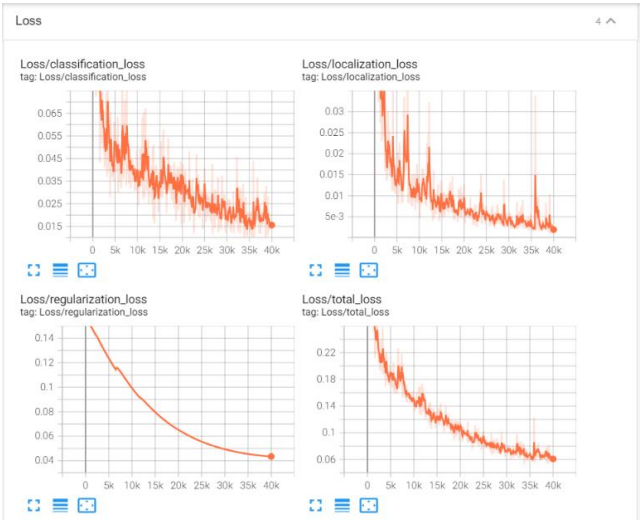


Fig. 13. Monitoring the training process using TensorBoard.

On the Raspberry Pi side, a Raspberry Pi compatible ARM build of Ubuntu 20.04. The operating system was mainly used since it comes with Python 3.8 as the default Python version. A Python version less than 3.10 is needed to install specific TensorFlow Lite runtime packages that the Coral TPU needs to run. This accelerates the process of development on Python while having the required dependencies for the Coral hardware. Another Ubuntu configuration was enabling the Bluetooth module and editing the ***systemd*** service configuration file for the Ubuntu Bluetooth stack to add the Serial Port Profile (SPP) - which is the profile of the HC-05 module - to the SDP (Service Discovery Protocol) database. The utility ***rfcomm*** was then added to the ***/etc/modules-load.d*** file to ensure that the “serial over Bluetooth” required utility is loaded during system boot. The following command was then used to mount the HC-05 as a **rfcomm0** device using its MAC address.

$ sudo rfcomm bind rfcomm0 <HC-05 MAC Address>

Another important configuration for Ubuntu was to set up an OpenSSH server to allow for remote access to the Raspberry Pi when mounted on the drone. Additionally, to allow Ubuntu to boot up without a monitor, the following two lines were added to the boot file ***config.txt***. The first line simulates the presence of an HDMI output device and the second enables HDMI mode with sound support.

hdmi_force_hotplug=1
hdmi_drive=2

One downside of using Ubuntu 20.04 is that it's very difficult to get the newer Raspberry Pi camera (Module 3) drivers to run on its kernel. There is no clear instruction on how this could be achieved.

The next part of the ***Software/Firmware*** sub-block is the Python code that performs object detection and sends commands to the ground vehicle based on the recognized objects. TensorFlow Lite libraries were used to forward the input frames from the camera to the trained model and read information about the detected object including its name, location, and confidence score. Upon the detection of the robot and a specific cone color, an ASCII letter is sent over the Bluetooth network. The ground vehicle is responsible for decoding these letters and executing tasks accordingly. An important feature of this implementation is that the Raspberry Pi is always sending what it "sees" regardless of whether or not it detected a known object or not. This makes the ground vehicle solely responsible for deciding when to listen or to ignore the incoming data on the Bluetooth serial channel. Two optional command line arguments are provided to the user to allow for more flexible and interactive code. The first argument allows the user to choose whether or not to activate the Coral TPU to speed up the process of detection. The second argument is a debug mode, which uses OpenCV features to display the camera feed along with useful information and visuals on each frame. These visuals include printing the frames per second (FPS) rate that the program is processing frames at and drawing boxes around the detected object with their corresponding detection confidence score. When run, the program always tries to bind the HC-05 MAC address to a rfcomm0 device using the bind command in the ***rfcomm*** Linux utility whether or not it was already bound. In other words, the command listed above is always executed at the beginning of the program. The code provides the option to record the flight from the perspective of the drone.

B. Ground Platform

The platform used is the TI Robotics System Learning Kit (RSLK) robot. This robot is designed to be a suitable platform for educational purposes, which provides a reasonable balance between building robotics systems and embedded design [7]. The kit provides a ready-to-use platform out of the box equipped with various sensors and actuators, which makes it a good choice to accelerate the process of designing robotic applications without extensive knowledge about the assembly aspect of robotics. The platform and its core microcontroller, the MSP432, are well documented by TI, which also provides online learning materials for the platform. This adds another reason why this platform was selected for this implementation of the framework. Programming the MSP microcontroller is done through the TI Code Composer Studio, which is built on the infamous Eclipse engine, which makes the development environment more familiar.

1) Hardware

The platform was brought up by connecting the necessary modules to the RSLK platform. This includes mounting the TI MSP432-based development board on the RSLK robot chassis, installing the motors and wheels, mounting the analog distance sensors on the front of the chassis, and connecting the HC-05 Bluetooth module to the appropriate pins. These pins are the UART TX/RX pins along with the power pins. This is because the HC-05 module is based on emulated serial behavior over Bluetooth [7].

The TI MSP432 is the main engine of the robot. The microcontroller provides many clocking sources and options, which eases the development of timers, UART configuration, interrupt generation, and many other tasks. The MSP432 provides multiple timer modules with different precisions, for example, the 16-bit Timer A module. The various timers on the MSP432 provide diverse operation and output modes, which accelerate tasks like generating PWM signals. Another important module on the MSP432 is the Enhanced Universal Synchronous/Asynchronous Communication Interface (EUSCI) module, which is adaptable for serial communication protocols such as UART, SPI, and I2C. The configurable baud rates, extensive interrupt support, and wide configurability of serial communication protocols ensure smooth serial communication development. The EUSCI was mainly used to send UART messages to the HC-05 Bluetooth module. Finally, the ADC module enables the conversion of analog signals into digital values. With configurable resolutions ranging from 8 to 16 bits, multiple input channels, and support for various sampling modes, the ADC allows for integrating the analog distance sensor in the implementation [8] [13] [15].

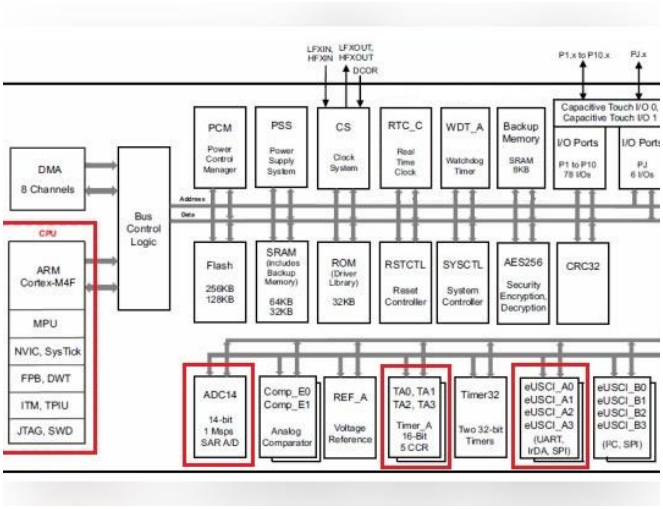


Fig. 14. Block diagram of the MSP432 microcontroller with the main parts of the implementation highlighted.

For the analog distance sensors, three sensors are used such that one is pointed to the right, center, and left. The main use of these sensors is to be able to detect when the robot arrives at the target object. This is mainly because the robot is responsible for handling all ground tasks based on the object detection command coming from the aerial vehicle in this initial implementation of the framework. The range of these sensors is about 10 to 80 centimeters, which makes it suitable for close-range tasks [4].

There is no native wireless communication support on the TI MSP432 development board, which is the reason why the HC-05 Bluetooth module was used. The module is easy to use, cost-effective, and small, which makes it a perfect pick for getting started with wireless communications. As mentioned before, the module relies on the UART communication protocol, which is fairly straightforward to implement on the TI microcontroller using the EUSCI communication module. The module provides about 10 meters of range, which is suitable for an initial implementation of the framework. The following figure shows the final form of the robot [8][9].

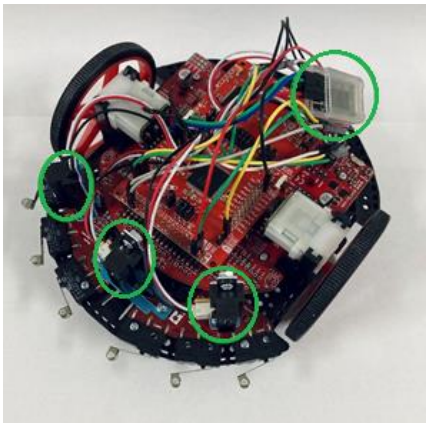


Fig. 15. The final form of the robot with the HC-05 module and the GP2Y0A21YK0F sensors highlighted.

2) Software/Firmware

The first step involving software was configuring the clocks to allow for having full control over the different clock sources when implementing different parts of the design. The clock configuration is mainly done using the Clock Source (CS) registers to configure the system clock to be 48MHz. Other clock sources are set up to allow the SMCLK to have a value of 12MHz. SMCLK is used as the source for both Timer A and EUSCI A2 modules. The CS registers are configured so that SMCLK and HSMCLK are sourced from HFXTCLK, MCLK is sourced from HFXTCLK, HSMCLK clock is divided by 2, and SMCLK clock is divided by 4. Following the clock tree in the datasheet shows that these configurations will result in the desired clock values [9]. The MSP432 clock tree is shown in the following figure.

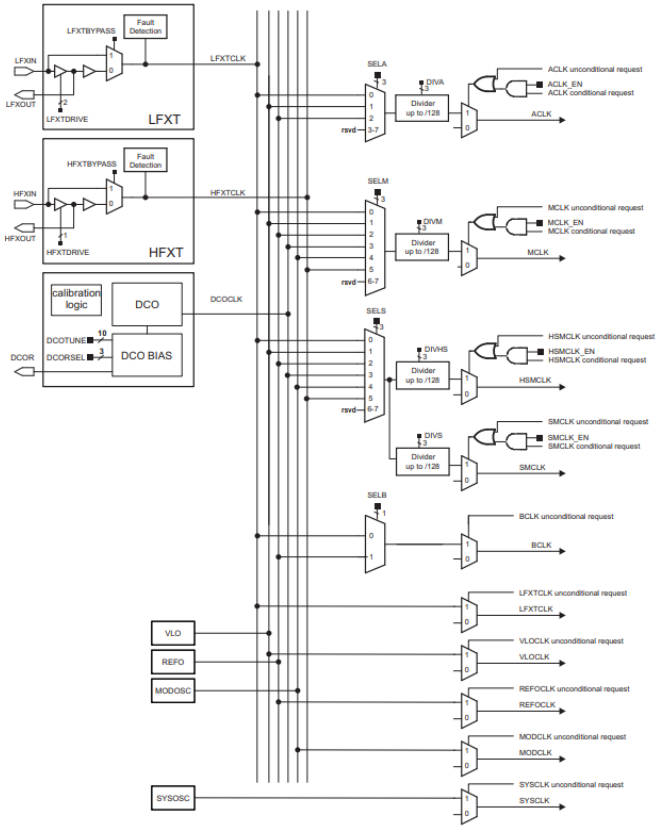


Fig. 16. MSP432 clock tree.

To generate PWM signals, Timer A0 was used. The initialization process of the timer included disabling interrupts, using the output modes of the timer to output the desired PWM signal, and using the timer in the up/down mode. The output mode Toggle/Reset is used with the up/down mode of the counter. The timer mode is different than the output mode/pattern. The Up/down mode of the timer used with the Toggle/Reset output mode is shown below [8][9].

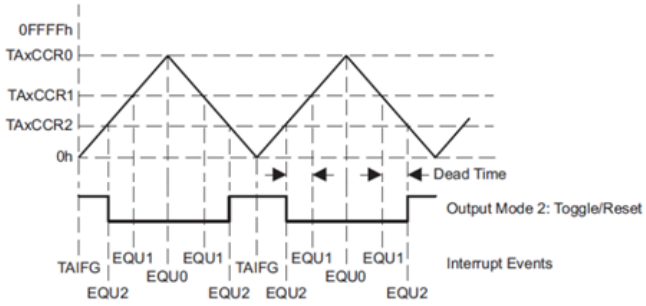


Fig. 17. Using up/down counter mode with toggle/reset output mode to generate the desired PWM signal.

Also, in the timer A0 initialization function, the pins P2.6, and P2.7 were set to the output of the timer. SMCLK (12MHz) is used as a source with a total division of 8, which results in a frequency of 1.5MHz. After that, the CCR0 (max. value) is set to the desired PWM period and the two compare/capture values associated with the pins above are set to an initial duty cycle state to get a result similar to Figure 4. The duty cycle can be updated using other functions. The duty cycle of the PWM signal driving the enable signal of the motor driver will control the speed of the robot.

When this timer is initialized, the CCR0 is set to 15,000, which results in a PWM signal with a frequency of 50Hz ($\frac{1500,000}{15,000*2}$) where the "2" represents the Up/Down mode of the timer. In the motor driver C file, different functions are implemented to control the movement of the robot by setting or clearing the different pins in the following diagram. This controls the direction of the wheels. The diagram below is generated based on how the TI DRV8838 motor driver is connected to the microcontroller.

TI-RSLK Chassis Board	MSP432 LaunchPad Pin	DRV8838	Description
DIRR	P5.5	PH	Right Motor Direction
nSLPR	P3.6	nSLEEP	Right Motor Sleep
PWMR	P2.6	EN	Right Motor PWM
DIRL	P5.4	PH	Left Motor Direction
nSLPL	P3.7	nSLEEP	Left Motor Sleep
PWML	P2.7	EN	Left Motor PWM

Fig. 18. The TI DRV8838 motor driver connection to the MSP432.

The second part of the ***Software/Firmware*** sub-block is configuring the EUSCI module to generate UART signals on the TX and RX pins. These pins are physically connected to the HC-05 RX and TX pins. The EUSCI A2 module is configured to be sourced from SMCLK (12MHz) with a baud rate division value of 104, which gives the closest number to the standard 115200 baud rate: $\frac{12000000}{104} = 115384$. The module is also configured to be in asynchronous mode with no 8-bit data, no parity bit, 1 stop bit, and the least significant bit first (little endian). The endianness was chosen to be little to match the ***pyserial*** library, which is used to send commands from the Raspberry Pi to the HC-05 module mounted on the robot chassis. The interrupts were enabled and silenced and the TX and RX pins were set to their primary function, which is the UART function. Two functions are defined to send a byte or read a byte from the TX/RX buffers.

The third part of the ***Software/Firmware*** sub-block is the ADC configuration to read the distance values in millimeters from the GP2Y0A21YK0F infrared distance sensors. A pre-written driver for the sensor was used including the configuration, filtering, and calibration of the sensor. The configuration process includes setting SMCLK (12MHz) as the source clock of the ADC, setting the conversion sequence mode to sequence-of-channels, and enabling automatic conversions. It also includes setting the start address of the memory where conversion results are stored, the resolution to 14 bits, and the power mode to regular. Multiple MCTLx (Memory Control) registers are also configured to set up the three channels used for the three sensors. The output of the ADC is filtered and calibrated to get the best distance results from the three sensors. An external driver library was used for that.

The distance sensors are used for two major reasons. The first use case is by stopping the robot when the sensors detect a very close distance from any of the three sides, left, center, and right. The second use case is for mitigating the error caused by imperfections in wheels, motors, and so on, which in turn causes the robot to not go straight. The two side sensors work on making the robot follow an object when an object is within a certain range. This helps reach the target object in case of any unexpected drifts.

The main program is a state machine that listens to the commands coming from the Raspberry Pi on the drone until it recognizes a command on which a task should be executed. The robot goes into the task execution state and ignores the commands coming from the drone until it's done executing that task. In this implementation of the framework, the robot goes into a delay before it starts listening again to commands from the drone. On-chassis RGB LEDs are used to show the state that the state machine is currently in.

IV. Testing

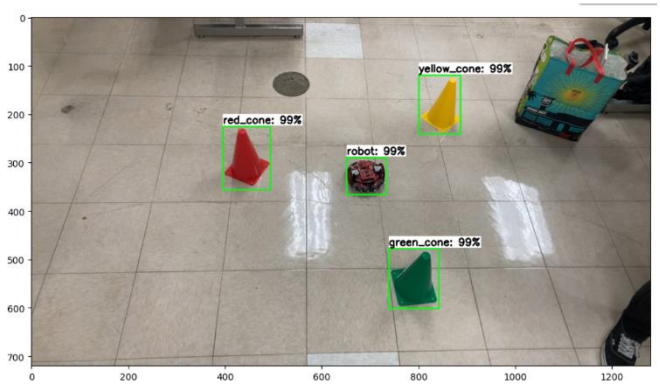
This section provides an overview of the different bring-up and testing stages for both platforms in the framework, ***Aerial Platform*** and ***Ground Platform***.

A. *Aerial Platform*

The first test for the aerial platform is to fly the drone. This was done with extra safety measurements including flying the drone at very low altitudes. The testing process included checking for stability, the response of the drone to changing throttle joystick levels, and different flight modes. Since the flight controller is a clone of the original Pixhawk 1 and since pre-arm safety checks were disabled, most autonomous flight modes weren't tested. In general, only modes and settings that are needed in this initial implementation of the framework were tested.

Fig. 19. Flying the drone at low altitude.

For the trained TensorFlow Lite model that will run on the Raspberry Pi, the model was first tested with static images that it wasn't exposed to before. This step was important to know if the model needs to be retrained with more images. The following figures show one of the test images using the ***float32***-weights model versus the ***int8***-weights model



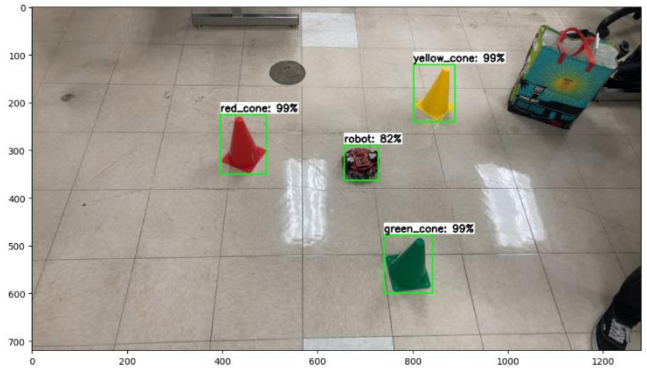


Fig. 20. Testing the trained models with new pictures of the targeted objects. (Top): float32 weights. (Bottom): int8 weights.

The next step was to test the model with a live feed from the Raspberry Pi camera. The purpose of this test was to test the model in a situation similar to the actual use case. Another reason was to get information on how fast the model detects objects, which was mainly measured by calculating the frames per second (FPS) and displaying it on the screen. For this test, a comparison was made between the base quantized model and the quantized model compiled for the Coral TPU. This proved a strong advantage for the Coral case.

Fig. 21. Object detection performance comparison (FPS) with and without Coral TPU. Coral on the bottom.

Another test on the Raspberry Pi side was to connect to the HC-05 through the serial terminal open-source tool ***minicom*** using the following two commands. The test consisted of sending diverse ASCII letters from the ***minicom*** terminal.

$ sudo rfcomm bind rfcomm0 <HC-05 MAC Address>

$ sudo minicom -d /dev/rfcomm0 -b 115200

The result of this test was verified from the TI MSP432 side and will be explained in the next section.

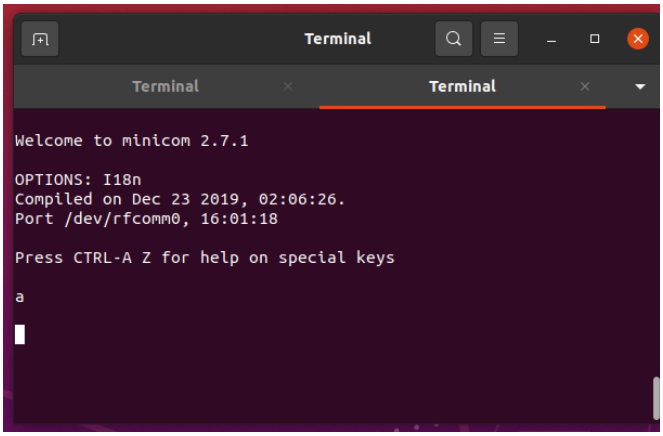


Fig. 22. Sending ASCII ‘a’ from minicom serial terminal.

B. *Ground Platform*

The first test for the TI robot was to make sure that the HC-05 could receive the ASCII character, in this case, ‘1’, sent from the Raspberry Pi. The Digilent Digital Discovery logic analyzer was used to detect and decode the UART transaction as shown in the following.

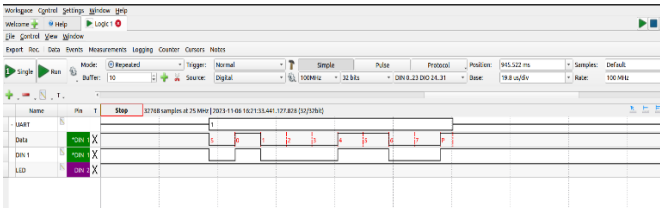


Fig. 23. Receiving the ASCII ‘1’ from the robot side.

Another test was to check that the three distance sensors driver returns correct values that are within the theoretical range of 10-80 cm. The values of the three sensors were printed on the serial terminal while inserting objects in front of it. The following figure shows the output of the terminal.

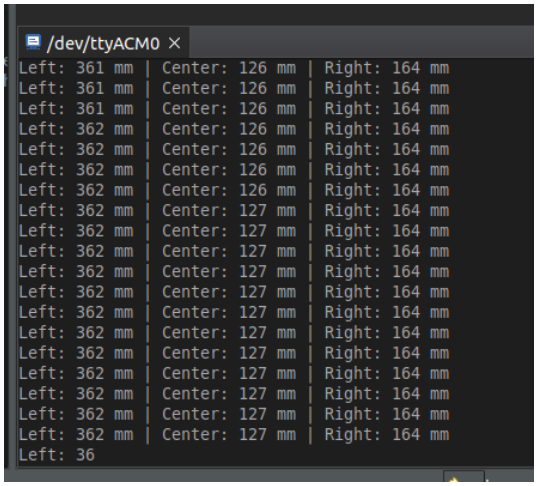


Fig. 24. The distance sensor readings in millimeters.

The last two tests were to check the main logic of the state machine running in the super-loop. This was done by sending different commands from the Raspberry Pi and observing the state changes on the TI robot side. Each state is given a unique color on the on-board RGB LED. Each state was also assigned to a certain movement like moving forward, backward, and sideways. These two tests allowed for verifying the functionality of the state machine as well as the PWM code.

C. *Overall Design*

The final test before the actual demo included integrating all blocks of the framework implementation. The drone was tested with all components and parts attached by flying it at low altitudes. Although the Raspberry Pi was mounted on the drone as shown in Fig. 7, it wasn’t powered up by the battery before it was individually tested. The Raspberry Pi, camera, and the AI model were tested by simulating the drone use case. This was done by running the Raspberry Pi with its battery to detect objects and send commands to the robot from a higher ground. The record option of the detection code was used to verify that what the drone “sees” is correct. Another part of the test environment was to power up the robot and verify that the state machine works as the camera detects different cones. To start the Python detection code on the Raspberry Pi, another personal computer was used to remotely access the Raspberry Pi using OpenSSH and execute the test code. The following figure shows some images taken by the Raspberry Pi camera during the flight simulation test.

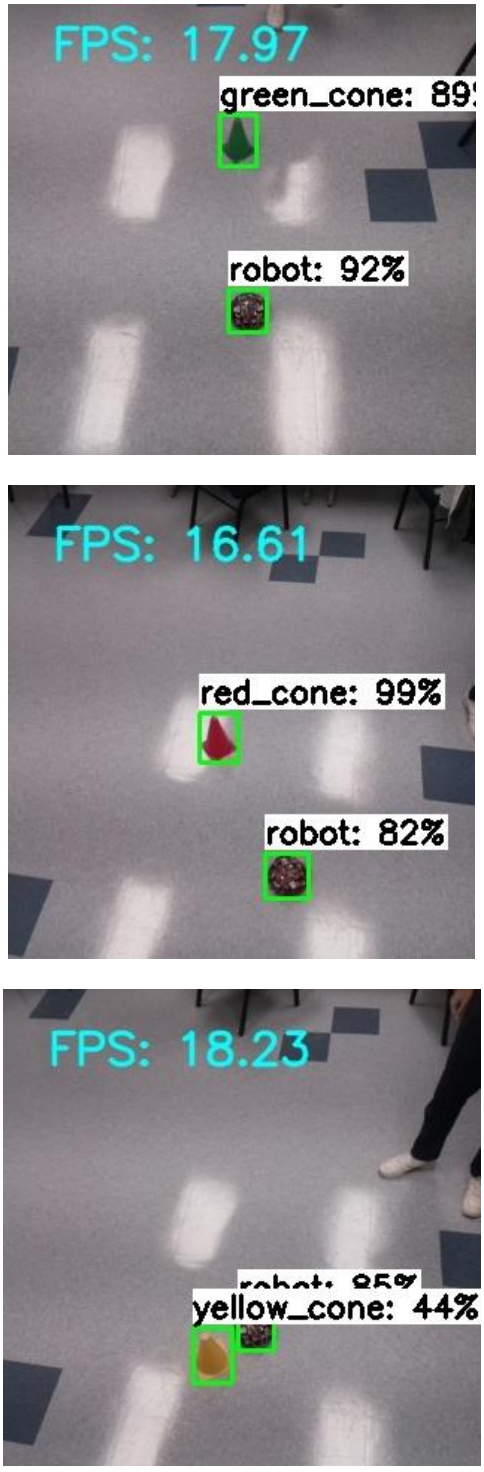


Fig. 25. Some images taken by the Raspberry Pi Camera from a high ground showing what the drone "sees".

V. Future Agenda

The purpose of this implementation is to set up a solid base of the framework where the major components are well-defined. Another important part of the implementation is to provide a working design. By doing this, focusing on improving individual blocks of the design becomes an easier and more focused task. This section provides suggestions on how the design could be improved to move closer to a more accurate emulation of NASA's Mars mission. The improvements can be divided into ***Aerial Platform***, ***Ground Platform***, and overall system improvements.

A. Aerial Platform

For the aerial platform, it would be a highly desired task to move away from the Pixhawk 1 clone flight controller to a more authentic device like Pixhawk 6C, for example. Another task that comes hand in hand with upgrading the flight controller is to upgrade the modules attached to it like GPS and compass, for example. This allows for a more stable design where the original libraries and documentation can be used for the parts. Also adding more modules to the flight controller would make a more robust design, for example, LiDAR or Radar. Another task that would improve the quality of the design, would be by adding an autonomous flying option. This can be done by directly interfacing with the flight controller - through for example UART – to load scripts that allow for flying autonomously. A good example of achieving this would be using the Navio2 flight controller, which is designed to interface well with the Raspberry Pi.

Another improvement to the design would be adding stereo vision, which would allow for calculating distances and adding much more detection capabilities to the aerial platform. This needs more than one camera. For that, the ArduCam Multi Camera Adapter Module can be used to attach more than one camera to the Raspberry Pi [10]. An alternative would be replacing the Raspberry Pi 4 with the new Raspberry Pi 5, which has native support for two CSI interfaces allowing for directly connecting two cameras. Also related to cameras, the last suggestion is to look into moving away from Ubuntu 20.04 for a newer kernel as the one in Ubuntu 22.04 or even Raspbian Bookworm. However, it should be kept in mind that for the Coral TPU to work, the 2.9.1 version of the ***tflite-runtime*** library must be used for stability reasons. This library is only compatible with Python 3.9 or less [3].

B. Ground Platform

On the robot side, it could be beneficial to move to a more complex platform with a more powerful computer, instead of the MSP432. This could be important if more demanding tasks were to be run on the robot side, which would require an operating system. An example of this would be aiding the robot platform with its own object detection model. In the case of sticking with the TI RSLK, freeRTOS could be used on the TI MSP432. Another improvement would be adding more sensors and actuators to the platform to allow for executing more complex tasks when receiving commands from the aerial platform.

C. Overall Design

For the overall system improvements. It is highly relevant to look into implementing a more efficient wireless communication protocol to communicate between the two platforms. The current Bluetooth communication channel can be used up to around 10 meters, which is not very efficient [6]. This could be done by using a LoRa system, which allows for more than 5-kilometers range of communication. Another option would be adding a Wi-Fi module to the robot platform. This would allow for the use of popular protocols like Message Queuing Telemetry Transport (MQTT) for wireless communication.

Another overall system improvement is enhancing the AI model. This can be easily done by adding more pictures and objects to the database and using that to train a suitable model [12]. Finally, to move closer to the NASA implementations, the open-source framework ***F'*** (fprime) designed by NASA can be incorporated. This framework was designed to build reliable and modular flight software. The framework can be built and used on different platforms including ARM-based platforms like the Raspberry Pi [14].

VI. Conclusion

This paper describes a Mars mission emulation framework. This framework features an aerial platform, a ground platform, and a wireless communication medium between the two. The job of the aerial platform is to scan the operation environment and command the ground platform to execute tasks. This is an emulation of the NASA Mars mission. This paper also describes an initial implementation of this framework. This includes the implementation of diverse engineering concepts including, drone bring-up, flight controller calibration, single-board computer integration, computer vision (AI) model training, embedded systems, robotics, data acquisition, driving actuators, and Bluetooth wireless communication. The paper also features some suggestions for future work to improve the framework implementation. In addition to emulating NASA's work, the project aims to expose students to concepts used in high-end space applications. The project also features the skill of high-level systems integration as well as the other in-depth technical skills described by the paper.

References

[1] "ardupilot_wiki/common/source/docs/common-pixhawk-overview.rst at master · ArduPilot/ardupilot_wiki," *GitHub*, Jul. 2023. https://github.com/ArduPilot/ardupilot_wiki/blob/master/common/source/docs/common-pixhawk-overview.rst (accessed Nov. 13, 2023).

[2] Evan, "Google Colaboratory," *colab.research.google.com*, Jan. 2023. https://colab.research.google.com/github/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/Train_TFLite2_Object_Detction_Model.ipynb#scrollTo=TI9iCCxoNlAL (accessed Nov. 14, 2023).

[3] "Get started with the USB Accelerator," *Coral*, Dec. 2020. https://coral.ai/docs/accelerator/get-started

[4] "GP2Y0A21YK0F." https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf (accessed Nov. 14, 2023).

[5] "HAWK'S WORK - YouTube," *www.youtube.com*, Sep. 2022. https://www.youtube.com/@hawkswork1911 (accessed Nov. 13, 2023).

[6] "HC-05 Bluetooth module," ITead Studio, 2010. Available: https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf

[7] J. W. Valvano, Embedded systems: introduction to robotics. 2020.

[8] "MSP432P401R, MSP432P401M MSP432P401R, MSP432P401M Mixed-Signal Microcontrollers 1 Device Overview," Texas Instruments, 2015. Available: https://www.ti.com/lit/ds/slas826e/slas826e.pdf

[9] "MSP432E4 SimpleLink™ Microcontrollers Technical Reference Manual," Texas Instruments, 2017. Accessed: Nov. 14, 2023. [Online]. Available: https://www.ti.com/lit/ug/slau723a/slau723a.pdf

[10] "Multi-Camera CamArray - Arducam Wiki," *docs.arducam.com*. https://docs.arducam.com/Raspberry-Pi-Camera/Multi-Camera-CamArray/Multi-Camera-CamArray/#arducam-multi-camera-adapter-board (accessed Nov. 14, 2023).

[11] S. Monk, *Raspberry Pi Cookbook*, 1st ed. "O'Reilly Media, Inc.," 2016.

[12] "tensorflow/models," *GitHub*, Nov. 2021. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[13] "Texas Instruments Robotics System Learning Kit User guide Robot Systems Learning Kit (TI-RSLK) User Guide 2 Texas Instruments Robotics System Learning Kit: The Solderless Maze Edition SEKP166," Texas Instruments. Accessed: Nov. 14, 2023. [Online]. Available: https://www.ti.com/lit/ml/sekp166/sekp166.pdf?ts=1699871498689&ref_url=https%253A%252F%252Fwww.google.com%252F

[14] "The Discerning User's Guide to F´," *F´*, 2017. https://nasa.github.io/fprime/UsersGuide/guide.html (accessed Nov. 14, 2023).