Task specification

Possible report structure

**Intro:**

**Project Overview:**

By meticulously planning and executing the following tasks, the engineering team will develop a functional prototype that effectively demonstrates the drone’s capabilities to partners. Emphasis should be placed on modularity, scalability, and demonstrable functionality to ensure a smooth transition from prototype to full-scale production.

**Final product:**

Develop a drone system designed to lift and park a 2,650 kg car using two collaborating drones, with LIDAR for SLAM, cameras and proximity sensors, BMS and powerful AI compatible control unit.

**Proof of concept:**

The prototype will consist of a single drone and a separate lifting mechanism to demonstrate individual functionalities for supervisors, collaborators and investors

**Software Engineering Tasks and Suggested Solutions**

*By Laszlo and Tobias*

**1. Control Algorithms Development**

* **Tasks:**
  + Develop motion control algorithms for navigation and lifting operations.
  + Implement coordination between sensors and actuators.
* **Processes:**
  + Use Python for high-level control on Raspberry Pi.
  + Implement real-time control tasks on microcontrollers using C.
* **Suggested Solutions:**
  + **Motion Control:** Use PID controllers for precise motor control.
  + **Libraries:** Utilize existing libraries like GPIO Zero and pigpio.

**2. Sensor Data Processing**

* **Tasks:**
  + Process data from LIDAR, cameras, and ultrasonic sensors.
  + Keep track of batteries and motors
  + Implement object detection and mapping.
* **Processes:**
  + Use machine learning models with the Hailo 8 AI accelerator for real-time processing.
  + Implement sensor fusion algorithms to combine data from multiple sensors.
* **Suggested Solutions:**
  + **Computer Vision:** Use OpenCV and TensorFlow Lite models optimized for Hailo 8.
  + **Mapping:** Implement SLAM (Simultaneous Localization and Mapping) algorithms for navigation.

**3. Microcontroller Programming**

* **Tasks:**
  + Integrate an operating system to handle all the tasks and information better
  + Integrate firmware for additional microcontrollers handling time-critical tasks.
* **Processes:**
  + Program in C++ and python using Visual Studio Code.
  + Establish communication protocols between microcontrollers and Raspberry Pi.
* **Suggested Solutions:**
  + **Robotic Operating System:**
  + **Real-Time Tasks:** Offload motor PWM control and lifting system controls to other microcontrollers
  + **Communication:** Use UART or SPI with error checking mechanisms. Additionally using Bluetooth connection as medium range communication.

**4. User Interface and Remote Monitoring**

* **Tasks:**
  + Integrate a user interface with live feed during operation for monitoring system status.
* **Processes:**
  + Create a screen streaming service for overseeing the operation of the drove with important data via Wi-Fi or Bluetooth.
  + Implement data logging for performance analysis. (Do we need it?)
* **Suggested Solutions:**
  + **Web Technologies:** Use Flask or Django for the server-side application.
  + **Bluetooth streaming:** Make a Bluetooth program for connecting, then streaming to a screen.
  + **Data Visualization:** Integrate libraries like Chart.js for real-time graphs.

*Because ROS2 allows us to stream data including images through LAN connection, the need for either web or Bluetooth connection and streaming is not needed.*

**5. System Integration and Testing (No experience with it yet)**

* **Tasks:**
  + Integrate software components and perform comprehensive testing.

*To simulate how the drone is supposed to work as close to realty as possible to minimize the time of hardware testing.*

* **Processes:**
  + Use unit testing frameworks to test individual modules.

*The implementation of a framework that goes through each module and testing them to see if they function as intended.*

* + Conduct field tests to validate system performance.

*Testing the drone with the hardware to see any misalignments and issues that may occur when switching from and ideal environment and fix them.*

* **Suggested Solutions:**
  + **Version Control:** Use Git for code management.

*The usage of Git was in consideration, but need was not there during the project period, but working without it was awkward, so the implementation of it will be done in the future.*

* + **Continuous Integration:** Set up CI/CD pipelines with tools like Jenkins.

*Because of the lack of experience with it and that the codes did not go through that many alterations when they were about to be completed the implementation of a CI/CD pipeline was not considered after its listing. Even though it is not really needed, the implementation of it sounds interesting after the project period, so its implementation is a possibility.*

*The simulation of the codes was done by software that are compatible with ROS2 by the names of rviz a 2d viewing program and gazebo which is a software that creates a virtual environment for the drone to move in.*

**6. Safety and Fail-Safe Mechanisms**

* **Tasks:**
  + Implement software safeguards to handle errors and unexpected conditions.

*To implement software safeguards ideas and plugins to looks for any potential errors and act accordingly.*

* **Processes:**
  + Develop error detection and recovery protocols.

*Create command lines to write out any errors that may have occurred and act upon them if necessary and protocols to recover from said errors if the is a possibility.*

* + Include watchdog timers to reset systems in case software hangs.

*To enable or implement a watchdog timer reset the system if the error that occurred was not recoverable.*

* **Suggested Solutions:**
  + **Action Report:** Every action when called must report back if the action was successful or not regarding given parameters and information.

*Since the easiest way to detect and potentially evade errors is to have a system that is non-stop talking to each other, meaning that when debugging, checking the testing logs could help in either narrowing down where the problem is or exactly pinpointing it.*

* + **Error Handling:** Use try-except blocks and logging for debugging.

*Every error should have a reaction from the program ranging from evasion or outright shutdown to not give the errors a chance to cause harm in anything.*

* + **Redundancy:** Implement redundant sensors for critical functions.

*One way of reducing the possibility of errors is having more sources of information even when it seems unnecessary, so that every portion of the mechanism can be overseen and with that much information even is an error occurs the program has a chance of reconstruction.*

**7. Documentation (No experience with it yet)**

* **Tasks:**
  + Document code, APIs, and user manuals.

*Documenting the code and its functionalities in a that it is understandable and good on the eyes.*

* **Processes:**
  + Write clear comments and use documentation generators.

*Writing clear and informative comments over code lines to help with understanding and remembering what does what. Implementing documentation generators to help with the documentation of the program with helpful tools.*

* **Suggested Solutions:**
  + **Tools:** Use Sphinx for Python documentation and Doxygen for C++ code.

*To be tried*