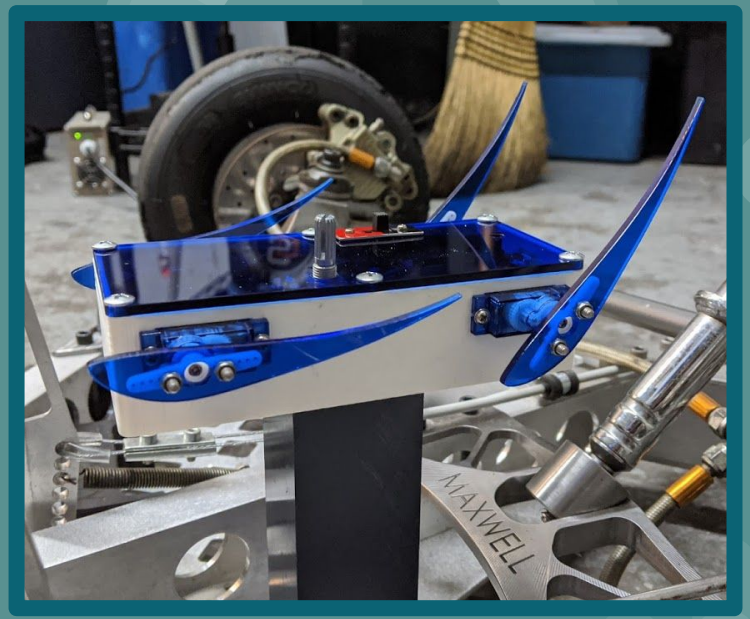
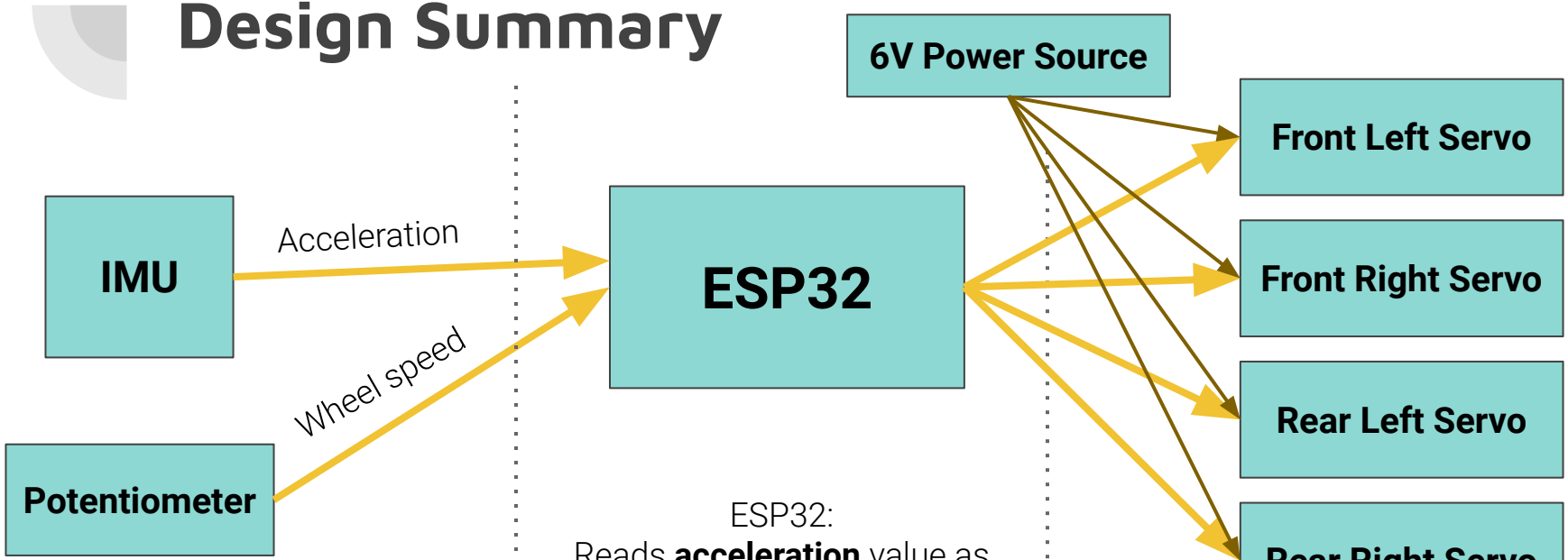
Active Race Car Aerodynamics

Joseph Macy and Miguel Nepomuceno

Link to Video



Design Summary



Input Sensors:
Detect vehicle acceleration and abstracts other car variables

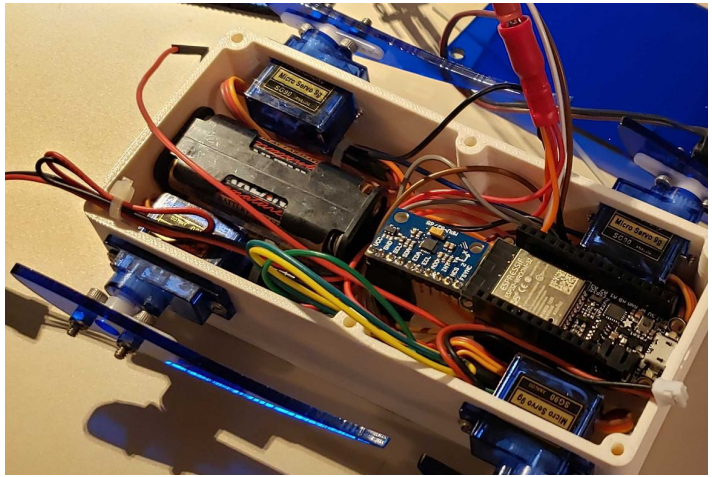
ESP32:
Reads **acceleration** value as key for lookup g-g table (aeromap), reading out **degrees** converted into **duty** values sent to servos

4 servo output allows each quadrant of the car to dynamically change angle of attack of flap element(s).

Sensing: Implementation

Main Sensor: **MPU9250**

- Used to measure 2D acceleration data
  - Potentially could improve responses by using 3D acceleration data or gyro readings
- Calibration: Sampled 100 data points at rest, and calculated a mean to offset raw data
- Tested output readings by doing numerous hand-tests before mounting onto the go-kart

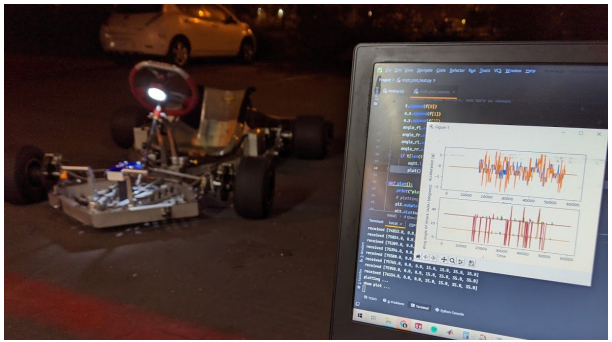


Sensing: Issues and Conclusions

- IMU was very sensitive to movement
  - Implemented **exponential smoothing**
- During hand-testing, any positive acceleration was shortly followed by a negative acceleration - it was hard to sustain high acceleration readings
  - **Could not test accuracy** of 2D lookup table. Final product visibly works well under simple 1D acceleration/deceleration
- Big Issue:
  - MPU9250 has a good resolution - **too** good for the ESP32.
  - ESP32 has memory allocation issues and cannot store values in a lookup table for every resolution point that the IMU can measure
  - Forced to have a comparably very low resolution for the possible usable 2D acceleration readings from the IMU

Actuation and Output: Implementation

- Primarily servo output, validated by observing that the sensor reacted as intended under cornering and straight acceleration
- Mounted a camera to the steering support of the go kart to observe the device in action.



MQTT Output:

- Sent lookup table degree output values through the MQTT server and received on local computer terminal.
- Graphed acceleration and servo angles in degrees with respect to time.

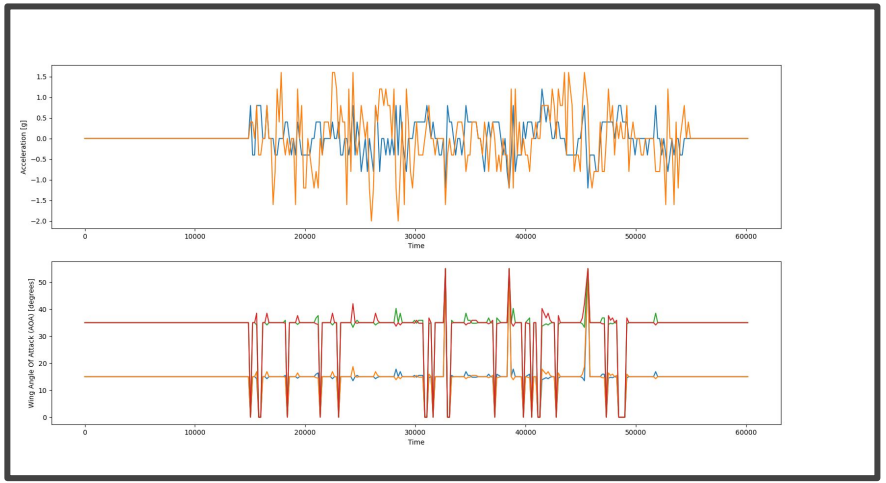
Actuation and Output: Issues and Conclusions

- Cheap servos had a large discrepancy between zero-duty states - resulted in some **rotational offset** between all four
- Duty cycles limited to between 30-100 running at 900 Hz
    - Had to create an equation mapping our full degree range (-5° to 55°) into that duty range. Hard to implement and **didn't get to make full range** of the resolution of our duty values
- Servos had poor rotational resolution
    - Also could not match up to the resolution of MPU9250
    - Additionally, some half-degree servo angle changes we made could make a big difference for aerodynamics, but were **hard to validate visually**.
- Otherwise, the actuation seemed to work extremely well under select circumstances. In the future would definitely use better servos.

Communication: Implementation

Main Protocol: **MQTT**

- ESP32 reads the IMU raw data, which is sent to the plotting host
- ESP32 finds duty values from lookup table, sends to MQTT and plotted versus time.
- MQTT was at least simple to use and easily interfaced between our network and the microcontroller
    - Easy to visualize thanks to the MQTT Explorer



Communication: Issues and Conclusions

- Our methods of collecting data were messy
  - **Hard to isolate cornering scenarios** when driving and also hard to window output accel data to identify different sorts of cornering
  - Not a clear distinction which direction the gokart was moving
  - However, easy to see that spikes in acceleration led to spikes in servo actuation
- In a realistic situation, MQTT probably would have sampling rate issues
  - However, since we were already decreasing resolution of IMU readings and throttling the speed at which readings came in to 0.1s, it did an okay job at reporting our data.
- Also will not always have stable connection to wifi or hotspot when driving. We might consider a more local protocol like bluetooth in the future

Computation: Implementation

Main Platform: **Micropython**

- Originally had branching code files (one for lookup table construction, one for sensor reading, one for MQTT data sampling and plotting)
    - Implemented construction code as a method called at the beginning of the file.
    - In reality, this **g-g diagram would not be generated this way**, it would be read off of much more complex simulation results.
- Used interrupts in order to end the MQTT data sampling time frame to transition into plotting.
- Implemented a **power switch** in order to save the four 1.5V batteries powering the servos from depleting.
- Overall code was not too complex (under 200 lines) and **saved processing power** on the ESP32

Computation: Issues and Conclusions

- Micropython on the ESP32 **does not support NumPy** - made table generation and basic acceleration vector mathematics more difficult
- Implementation of lookup table in Micropython took **a lot of memory**.
    - No matrix operations that would make this implementation easier. In reality, the g-g lookup stored on the ESP32 would be a sampled array of values from aero simulation and ultimately take less space
- Overall, Micropython was not a bad choice, but we would be curious to try out the Arduino implementation.

Conclusions

What worked:

- Got **fast, responsive movement** from the servos when we needed it
- MQTT data plotting worked smoothly and **all components were communicating** with each other properly.

What we would fix:

- Using higher quality servos and microprocessor to handle larger resolution lookup tables and execute those angles
- More intentional testing and MQTT outputs to identify performance during corners
- Using proper aero map for lookup table (not generating on ESP32) which is a more realistic model, and saves memory and computation