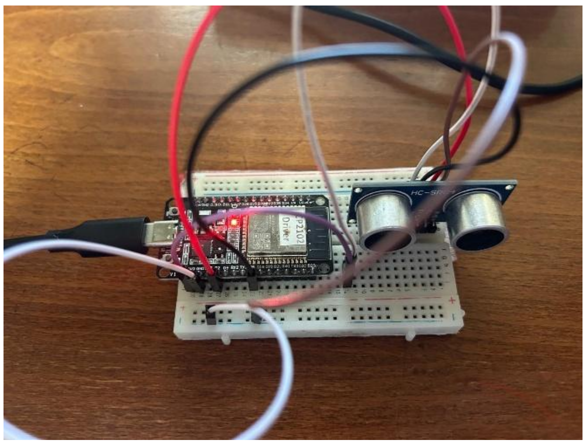
User Manual

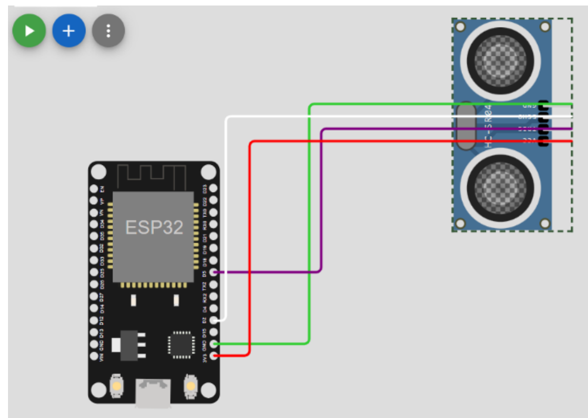
We'll have a walkthrough on how to run the whole pipeline:

For the distance sensor- **HC-SR04 with ESP32**:

1) Breadboard setup:

Including all the relevant wiring and hardware.

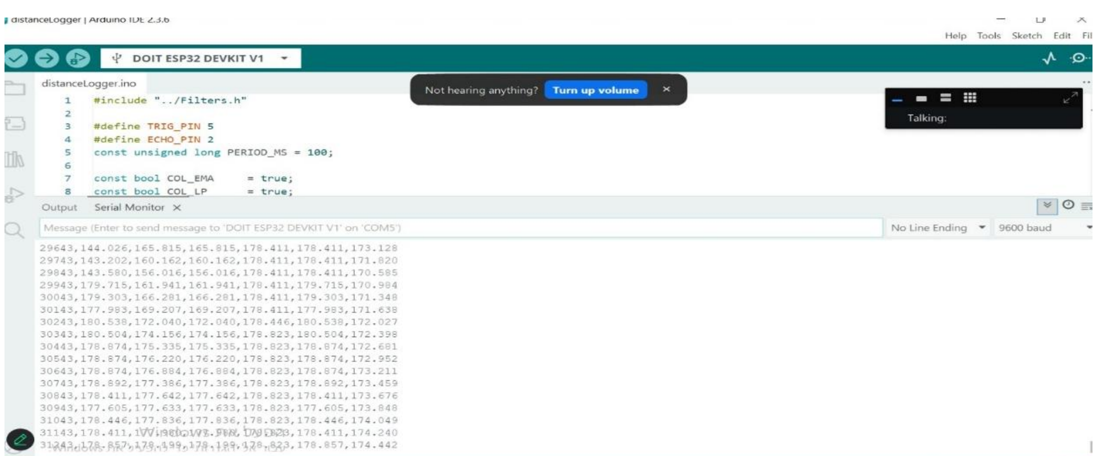




2) Uploading the ino sketch:

After connecting the wires we'll upload the distanceLogger.ino sketch using the Arduino IDE (with having the Filter.h file in the same directory) and then check the serial monitor to then go to step 3 in the pipeline. What you should see is as follows where the columns represent:

t_ms, raw_cm, ema_cm, lp_cm, median_cm, outlier_cm, kalman_cm



3) Collecting the data streams from the ESP32:

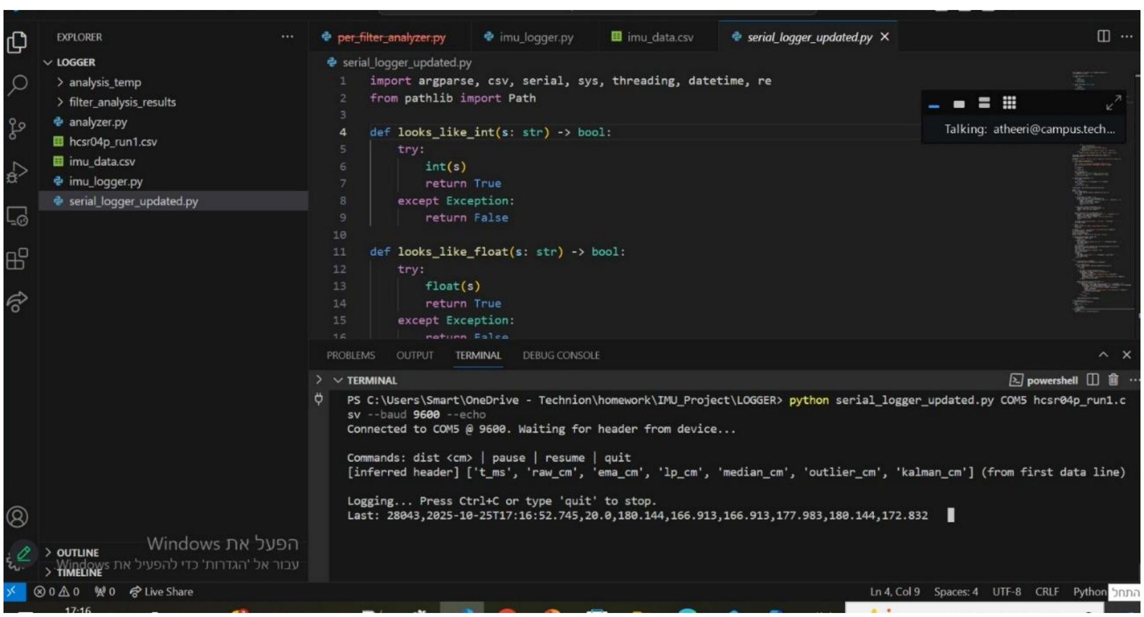
In the tools directory under the relative path: ./tools/HC-SR04P_tools directory you'll find a file that's called serial_logger_updated.py which reads real-time distance values sent from the ESP32 readings through the serial port and saves them into a CSV file with timestamps. Each entry represents one measurement in centimeters, allowing you to record how the distance changes over time.

To run it run the following command:

For instance on COM3 (if it's what is connected)-

```
python serial_logger_updated.py COM3 output.csv
```

What you'll see is:



4) Running the analyzer:

After running collecting the csv data files we'll run the analyzer code for getting the relevant plots/metrics and the full analysis where we'll get the following directory structure for the per filter analysis:

The script per_filter_analyzer.py analyzes and compares the performance of all filters (EMA, LP, Median, Outlier, Kalman) applied to your relevant datasets.

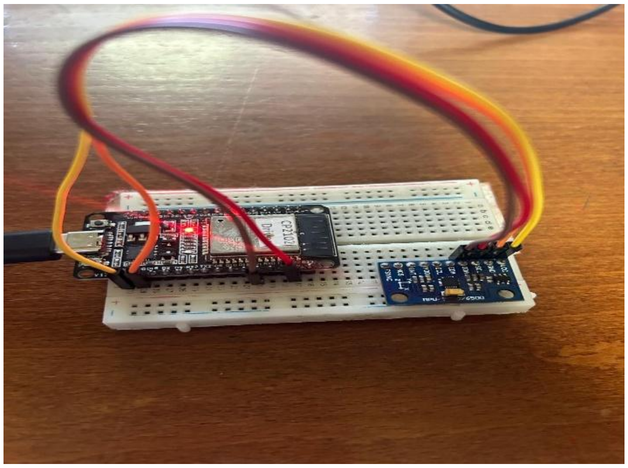
Each dataset (like distorted_20.csv, distorted_40.csv...) already contains the raw and filtered columns generated during logging. The analyzer loads these files, computes performance metrics for each filter, and produces graphs and summary CSVs automatically.

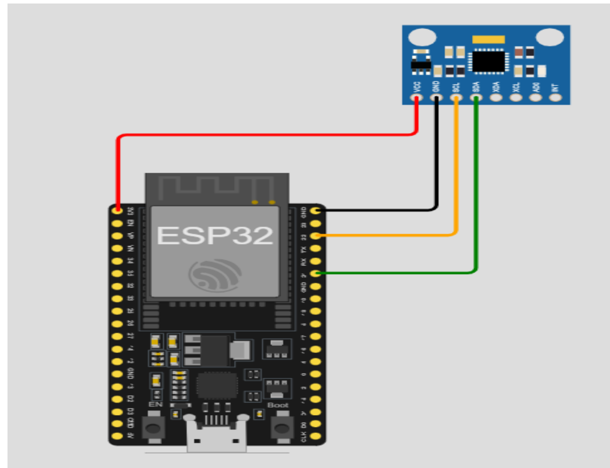
```
filter_analysis_output/
│
├── all_filters_summary.csv
├── rmse_comparison.csv
├── mean_comparison.csv
├── std_comparison.csv
├── rmse_comparison.png
│
├── ema/
│   ├── ema_comparison.png
│   ├── ema_metrics.csv
│   ├── distorted_20_ema.png
│   ├── distorted_40_ema.png
│   └── ...
│
├── lp/
│   ├── lp_comparison.png
│   ├── lp_metrics.csv
│   ├── distorted_20_lp.png
│   ├── distorted_40_lp.png
│   └── ...
│
├── median/
│   ├── median_comparison.png
│   ├── median_metrics.csv
│   ├── distorted_20_median.png
│   ├── distorted_40_median.png
│   └── ...
```

This is how the whole process works we'll now also explain how to process goes for the MPU6500 sensor.

For the IMU, **MPU6500** sensor:

1) Breadboard:





2) Uploading the ino sketch:

After connecting the wires we'll upload the distanceLogger.ino sketch using the Arduino IDE (with having the header files relevant to the IMU code in the same directory) and then check the serial monitor to then go to step 3 in the pipeline. What you should see is as follows where the columns represent:

time_ms, ax, ay, az, gx, gy, gz, ax_ema, ay_ema, az_ema,...

3) Collecting the data from ESP32:

In the tools directory under the relative path:

./tools/IMU_tools

you'll find a file called imu_logger.py which continuously reads real-time IMU data (accelerometer, gyroscope, and magnetometer values) sent from the ESP32 through the serial port and saves them into a CSV file. Each entry includes a timestamp and the corresponding sensor readings on all three axes (X, Y, Z), allowing you to record how the motion and orientation change over time.

To run it, use the following command:

For instance, if your ESP32 is connected on COM3 –

python imu_logger.py COM3 imu_data.csv

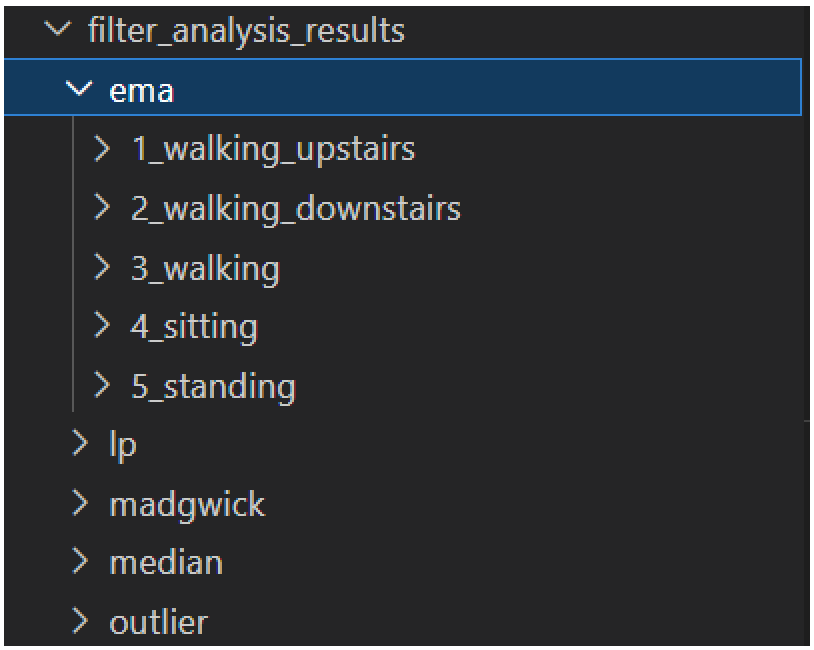
4) Running the analyzer:

After collecting the IMU CSV data files, we'll run the analyzer script to generate the relevant plots, performance metrics, and detailed analysis results.
The script filter_analysis_MPU6500.py analyzes and compares the performance of all filters (EMA, Median, Low-Pass, Outlier, and Madgwick) applied to the IMU readings.
Each dataset (for example: experiment_1.csv, experiment_2.csv, etc.) already contains the raw and filtered accelerometer and gyroscope signals recorded during data collection. The analyzer loads these files, calculates various performance metrics for each filter

(such as SNR, RMSE, smoothness, lag, and frequency distortion), and automatically produces graphs, per-activity visualizations, and summary CSVs for all filters. To run the analyzer, use the following command inside your project directory:

python filter_analysis_MPU6500.py

And the following directory structure will be generated:



So each filter would have his own directory and the experiments as directories so we can compare the raw data vs. the filtered data more profoundly.

That's basically how our pipeline works.