

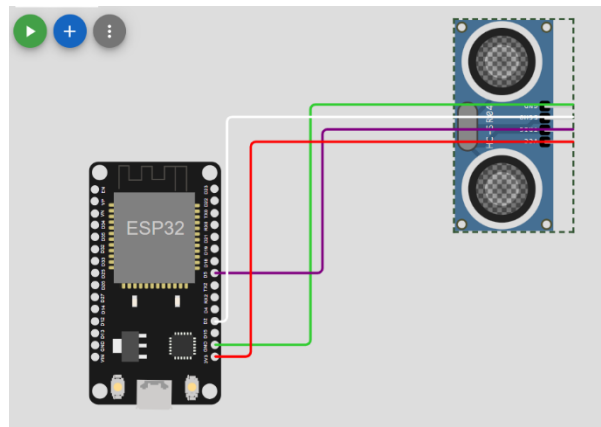
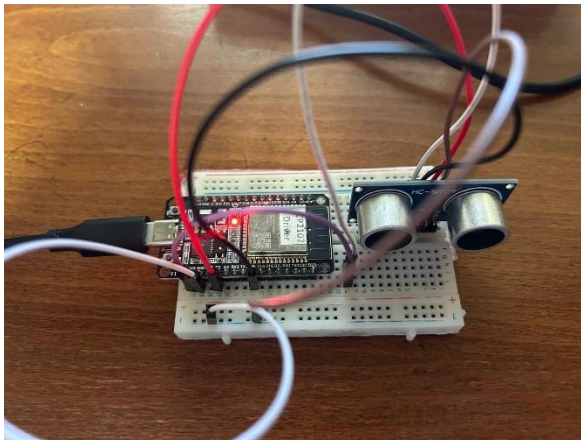
User Manual

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

```
distanceLogger.ino
1 #include "../Filters.h"
2
3 #define TRIG_PIN 5
4 #define ECHO_PIN 2
5 const unsigned long PERIOD_MS = 100;
6
7 const bool COL_EMA = true;
8 const bool COL_LP = true;
```

Output: Serial Monitor X

Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

No Line Ending 9600 baud

29643,144.026,165.815,165.815,178.411,178.411,173.128
29743,143.202,160.162,160.162,178.411,178.411,171.820
29843,143.580,156.016,156.016,178.411,178.411,170.585
29943,179.715,161.941,161.941,178.411,179.715,170.994
30043,179.303,166.281,166.281,178.411,179.303,171.348
30143,177.993,169.207,169.207,178.411,177.993,171.639
30243,180.538,172.040,172.040,178.446,180.538,172.027
30343,180.504,174.156,174.156,178.823,180.504,172.398
30443,178.874,175.335,175.335,178.823,178.874,172.681
30543,178.874,176.220,176.220,178.823,178.874,172.952
30643,178.874,176.884,176.884,178.823,178.874,173.211
30743,178.892,177.386,177.386,178.823,178.892,173.459
30843,178.411,177.642,177.642,178.823,178.411,173.676
30943,177.605,177.633,177.633,178.823,177.605,173.848
31043,178.446,177.836,177.836,178.823,178.446,174.049
31143,178.411,177.836,177.836,178.823,178.411,174.240
31243,178.446,177.836,177.836,178.823,178.446,174.442

3) Collecting the data streams from the ESP32:

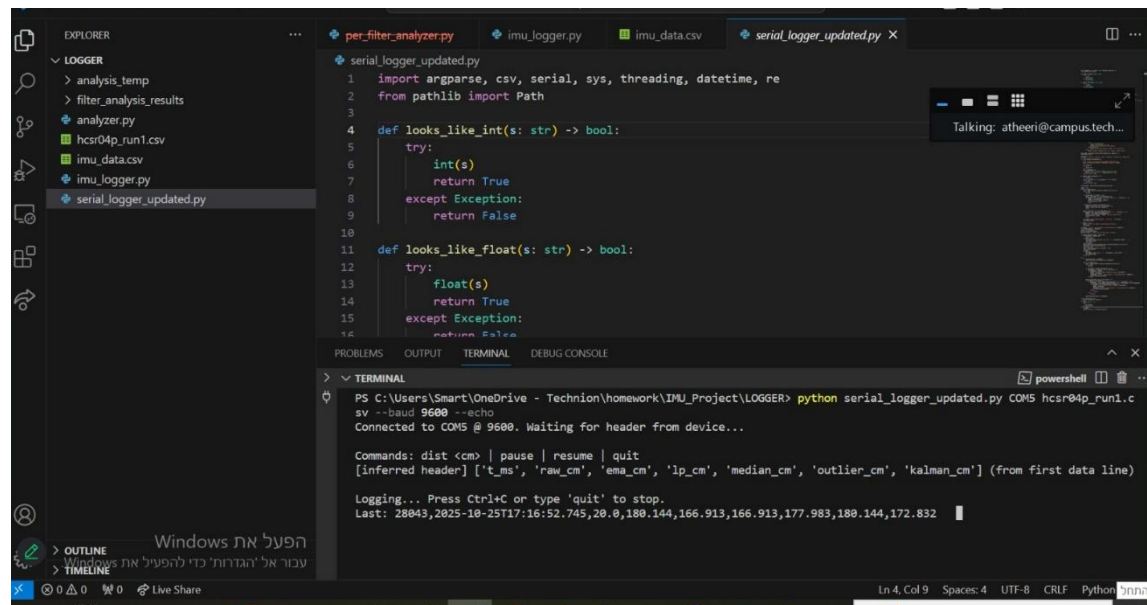
In the tools directory under the relative path: ./tools/HC-SR04P_tools directory you'll find a file that's called serial_logger_updated.py which reads real-time distance values sent from the ESP32 readings through the serial port and saves them into a CSV file with timestamps. Each entry represents one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



The screenshot shows a Visual Studio Code editor with the file explorer on the left displaying the project structure. The file explorer shows a 'LOGGER' directory containing 'analysis_temp', 'filter_analysis_results', 'analyzer.py', 'hcsr04p_run1.csv', 'imu_data.csv', 'imu_logger.py', and 'serial_logger_updated.py'. The main editor window displays the code for 'serial_logger_updated.py', which includes imports for 'argparse', 'csv', 'serial', 'sys', 'threading', 'datetime', and 're', and a 'Path' object from 'pathlib'. The code defines two functions: 'looks_like_int(s: str) -> bool' and 'looks_like_float(s: str) -> bool', both using try-except blocks to validate input. The terminal at the bottom shows the command 'python serial_logger_updated.py COM5 hcsr04p_run1.csv' being executed, with output indicating connection to COM5 at 9600 baud and waiting for a header. It then shows the inferred header ['t_ms', 'raw_cm', 'ema_cm', 'lp_cm', 'median_cm', 'outlier_cm', 'kalman_cm'] and the start of logging data, with the last line showing a timestamp and a list of distance measurements: 'Last: 28043,2025-10-25T17:16:52.745,20.0,180.144,166.913,177.983,180.144,172.832'.

```
1 import argparse, csv, serial, sys, threading, datetime, re
2 from pathlib import Path
3
4 def looks_like_int(s: str) -> bool:
5     try:
6         int(s)
7         return True
8     except Exception:
9         return False
10
11 def looks_like_float(s: str) -> bool:
12     try:
13         float(s)
14         return True
15     except Exception:
16         return False
```

```
PS C:\Users\Smart\OneDrive - Technion\homework\IMU_Project\LOGGER> python serial_logger_updated.py COM5 hcsr04p_run1.c
sv --baud 9600 --echo
Connected to COM5 @ 9600. Waiting for header from device...

Commands: dist <cm> | pause | resume | quit
[inferred header] ['t_ms', 'raw_cm', 'ema_cm', 'lp_cm', 'median_cm', 'outlier_cm', 'kalman_cm'] (from first data line)
Logging... Press Ctrl+C or type 'quit' to stop.
Last: 28043,2025-10-25T17:16:52.745,20.0,180.144,166.913,177.983,180.144,172.832
```

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.

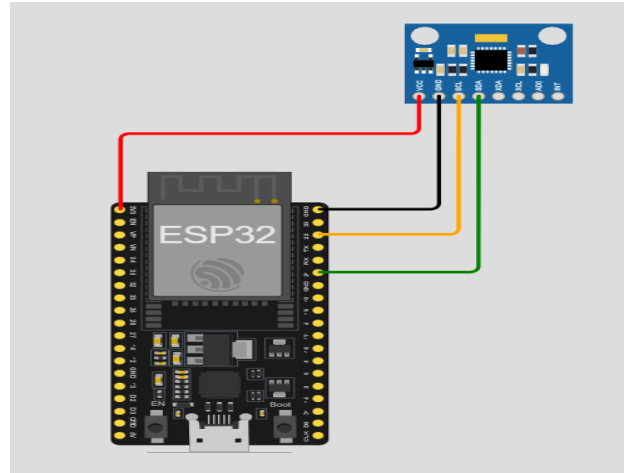
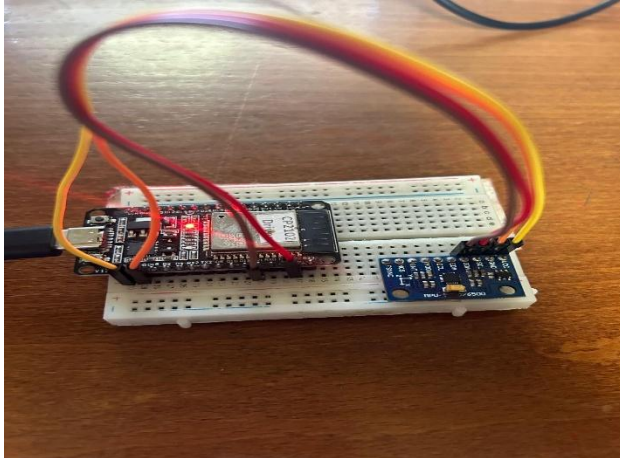
Each dataset (like `distorted_20.csv`, `distorted_40.csv`...) already contains the raw and filtered columns generated during logging. The analyzer loads these files, computes performance metrics for each filter, and produces graphs and summary CSVs automatically.

```
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,...

```
imu_logger.ino  Filters_IMU.h  MadgwickAHRS.h
219 float az = g2ms2(gValue.z);
220
221 // Gyro in deg/s -> rad/s
222 xyzFloat dps = myMPU9250.getDpsValues();
223 float gx = dps.x;
224 float gy = dps.y;
225 float gz = dps.z;
226
227 // dt from elapsed time
228 uint32_t t_now_ms = millis();
229 float dt_s = (t_now_ms - t_prev_ms) * 1e-3f;
230 t_prev_ms = t_now_ms;
```

Output Serial Monitor X

Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM5')

No Line Ending 115200 baud

206466,0.000000,0.000000,9.798861,0.019256,-0.001251,0.004379,0.000000,0.000000,9.807788,0.027443,-0.023725,-0.004140,0.000000,0.000000,9.807839,0.011627,-0.07496,0.000000,0.000000,9.800058,-0.011262,-0.062256,-0.018509,0.000000,0.000000,9.806242,0.019702,-0.031438,-0.007014,0.000000,0.000000,9.800058,0.011627,208526,0.000000,0.000000,9.804248,0.003997,-0.008881,0.004379,0.000000,0.000000,9.805843,0.016561,-0.026926,-0.004735,0.000000,0.000000,9.800058,0.003997,-209556,0.000000,0.000000,9.805656,0.011627,0.006378,-0.026138,0.000000,0.000000,9.804806,0.015574,-0.020265,-0.009016,0.000000,0.000000,9.800656,0.003997,-210586,0.000000,0.000000,9.809036,0.057403,-0.001251,0.012009,0.000000,0.000000,9.805652,0.023940,-0.016463,-0.004811,0.000000,0.000000,9.804248,0.011627,-211616,0.000000,0.000000,9.802452,0.072662,0.006378,-0.018509,0.000000,0.000000,9.805012,0.033684,-0.011894,-0.007551,0.000000,0.000000,9.802452,0.057403,-212646,0.000000,0.000000,9.808437,0.034515,-0.047028,0.004379,0.000000,0.000000,9.805696,0.033850,-0.018921,-0.005165,0.000000,0.000000,9.808437,0.057403,-213676,0.000000,0.000000,9.807240,0.026885,-0.062266,0.012009,0.000000,0.000000,9.806005,0.032457,-0.027594,-0.001730,0.000000,0.000000,9.807240,0.034515,-214706,0.000000,0.000000,9.793474,0.103179,0.097931,-0.010879,0.000000,0.000000,9.803499,0.046602,-0.002489,-0.003560,0.000000,0.000000,9.807240,0.034515,-215736,0.000000,0.000000,9.815022,0.065032,-0.001251,-0.018509,0.000000,0.000000,9.805803,0.050288,-0.002242,-0.006550,0.000000,0.000000,9.807240,0.065032,-216766,0.000000,0.000000,9.815212,0.057403,-0.024139,-0.010879,0.000000,0.000000,9.808485,0.051711,-0.006621,-0.007416,0.000000,0.000000,9.815022,0.065032,

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:

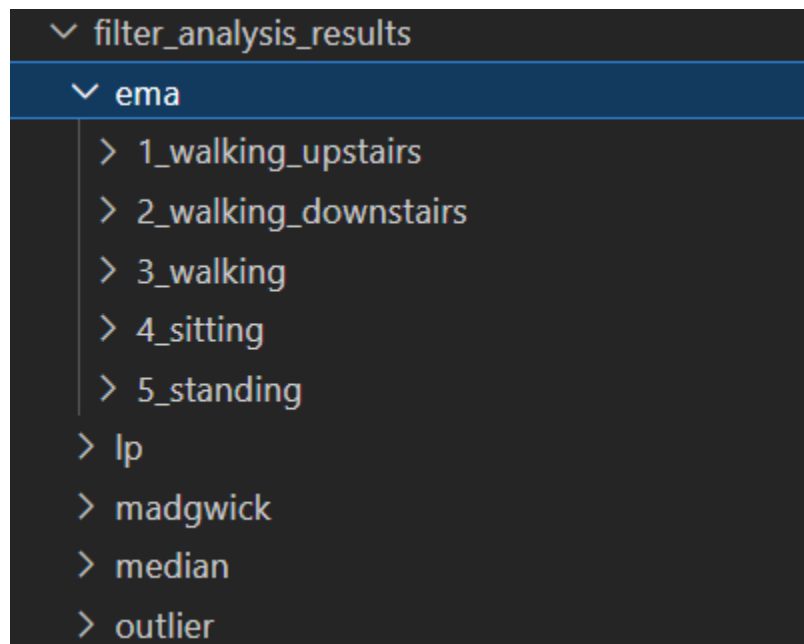
After collecting the IMU CSV data files, we'll run the analyzer script to generate the relevant plots, performance metrics, and detailed analysis results. The script `filter_analysis_MPU6500.py` analyzes and compares the performance of all filters (EMA, Median, Low-Pass, Outlier, and Madgwick) applied 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.