CLEMSON UNIVERSITY INTERNATIONAL CENTER FOR AUTOMOTIVE RESEARCH

AuE 8350

Automotive Electronics Integration

Project 1 Report

Fall 2021

Submitted By:
Shubham Gupta
C38771108
gupta9@clemson.edu

Project 1: Sensing Signal Processing

Task 1: Signal Processing for One Ultrasonic Sensor

1.1 Problem Statement

The Task 1 aims to implement a Kalman filter using an Arduino microcontroller to filter the noise from an ultrasonic sensor reading. In general, an ultrasonic sensor is used to measure distance of an object in front of it. While the ultrasonic sensor can be used without a filter, the obtained measurements are noisy, and we obtain fluctuations in measurements for the same distance. Therefore, a simple yet powerful filter like the Kalman filter is implemented in this project.

To achieve the above goal, the following steps need to be fulfilled:

- **Sampling**: Sample the time data of an ultrasonic sensor as quickly as possible
- **Filtering**: Implement a Kalman filter to filter out the noise from the readings
- **Calibration**: Identify an appropriate function to map the measured time values to the corresponding distance values

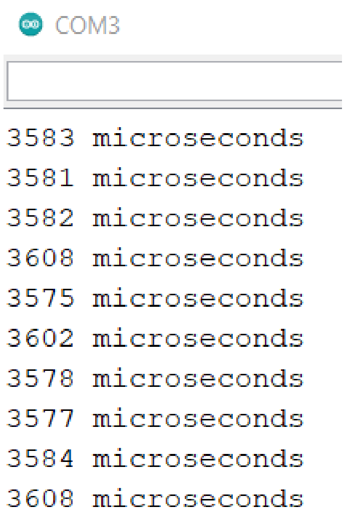
Furthermore, the following performance metrics need to be met for the Kalman filter:

- Converge the Kalman filter as quickly as possible (less than 10 seconds)
- Accuracy of calculated distance must be within +/-3 mm

1.2 Technical Approach

The task at hand was fulfilled by completing the steps highlighted in the problem statement as follows:

1. **Sampling**
   Time readings were sampled from the ultrasonic sensor as fast as possible.
    - For our project, chosen sampling time = 10 milliseconds
    - Note that these measured time values are noisy and need to be filtered to obtain a reliable time value. The following snippet shows time values for the same distance:



```
3583 microseconds
3581 microseconds
3582 microseconds
3608 microseconds
3575 microseconds
3602 microseconds
3578 microseconds
3577 microseconds
3584 microseconds
3608 microseconds
```

2. Kalman Filter

To implement a Kalman filter for one ultrasonic sensor, the following Prediction and Correction equations were used:

Process Model:

$$y_k = 1*y_{k-1}$$
$$z_k = 1*y_k + v_k$$

where:

- y: system state (time values)
- k: discrete time value
- z: measured data (time measurement from sensor)
- v: measurement noise

Prediction	Correction
$\hat{y}^-_k = y_{k-1}$ $P^-_k = P_{k-1}$ where: • $\hat{y}^-_k$: Predicted value based on previous measurement • P^-_k : Predicted error covariance	$K = P^-_k (P^-_k + R)^{-1}$ $\hat{y}_k = \hat{y}^-_k + K(z_k - \hat{y}^-_k)$ $P_k = (I - K) P^-_k$ where: • K: Kalman Gain • $\hat{y}_k$: Estimated State • P_k : Estimated error covariance

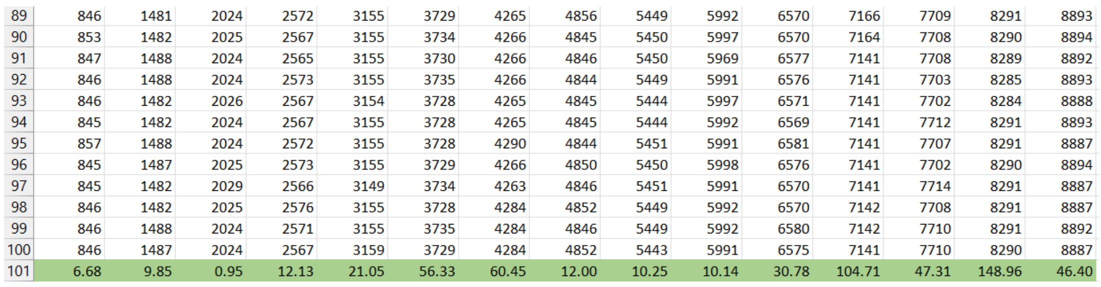
i) Error Covariance R

To implement Kalman filter using above equations, we first need to determine the error covariance R. This was determined by taking 100 samples each at different distances from the sensor within the **CU-ICAR auditorium** and calculating the covariance within those measurements. The calculated error covariances were then included in our codes for Kalman filter.

The code for calculating R is as follows:

```
// Calculating R (measurement covariance) by taking sample measurements
int numSamples = 100;                       // Number of samples taken to calculate R
float duration[numSamples];             // Array to store sample time duration values
float durSum = 0;                       // Sum of time durations (to calculate mean)
for(int i=0; i<numSamples; i++){
  triggerSensor();
  duration[i] = pulseIn(echoPin, HIGH);
  durSum += duration[i];
  Serial.println(duration[i], 2);
  delay(10);
}
float durMean = durSum / numSamples;        // Calculating mean of time durations
float durVar = 0;                           // Helper variable to calculate variance
for(int i=0; i<numSamples; i++){
  durVar += (duration[i] - durMean)*(duration[i] - durMean);
}
float measure_variance = durVar / numSamples;      // Calculated duration variance
```

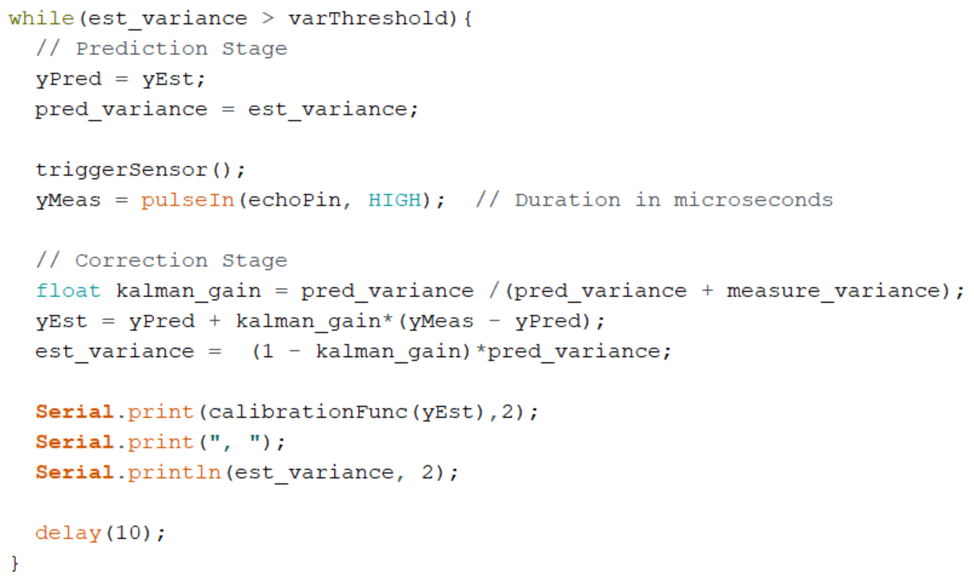
We calculated R values for distances 10cm to 150cm in intervals of 10cm each. A snippet of the collected data can be seen below:



89	846	1481	2024	2572	3155	3729	4265	4856	5449	5992	6570	7166	7709	8291	8893
90	853	1482	2025	2567	3155	3734	4266	4845	5450	5997	6570	7164	7708	8290	8894
91	847	1488	2024	2565	3155	3730	4266	4846	5450	5969	6577	7141	7708	8289	8892
92	846	1488	2024	2573	3155	3735	4266	4844	5449	5991	6576	7141	7703	8285	8893
93	846	1482	2026	2567	3154	3728	4265	4845	5444	5997	6571	7141	7702	8284	8888
94	845	1482	2024	2567	3155	3728	4265	4845	5444	5992	6569	7141	7712	8291	8893
95	857	1488	2024	2572	3155	3728	4290	4844	5451	5991	6581	7141	7707	8291	8887
96	845	1487	2025	2573	3155	3729	4266	4850	5450	5998	6576	7141	7702	8290	8894
97	845	1482	2029	2566	3149	3734	4263	4846	5451	5991	6570	7141	7714	8291	8887
98	846	1482	2025	2576	3155	3728	4284	4852	5449	5992	6570	7142	7708	8291	8887
99	846	1488	2024	2571	3155	3735	4284	4846	5449	5992	6580	7142	7710	8291	8892
100	846	1487	2024	2567	3159	3729	4284	4852	5443	5991	6575	7141	7710	8290	8887
101	6.68	9.85	0.95	12.13	21.05	56.33	60.45	12.00	10.25	10.14	30.78	104.71	47.31	148.96	46.40

ii) **Code for Kalman Filter**

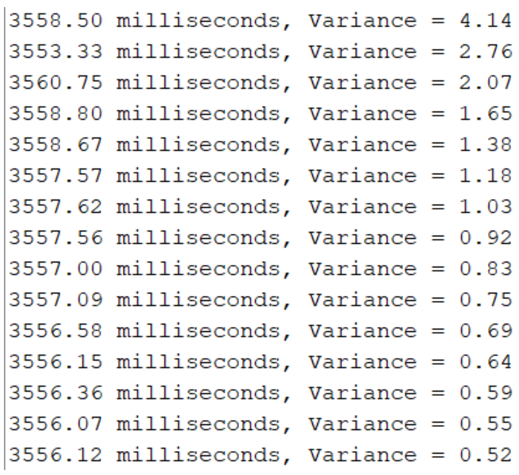
The Kalman filter was implemented on the Arduino using the following code:



```
while(est_variance > varThreshold){
  // Prediction Stage
  yPred = yEst;
  pred_variance = est_variance;

  triggerSensor();
  yMeas = pulseIn(echoPin, HIGH);  // Duration in microseconds

  // Correction Stage
  float kalman_gain = pred_variance /(pred_variance + measure_variance);
  yEst = yPred + kalman_gain*(yMeas - yPred);
  est_variance =  (1 - kalman_gain)*pred_variance;

  Serial.print(calibrationFunc(yEst),2);
  Serial.print(", ");
  Serial.println(est_variance, 2);

  delay(10);
}
```

Here, the threshold variance to stop the Kalman filter convergence was taken = 0.07

The time values obtained with Kalman filtering are as follows. As seen, they start converging towards a reliable value and the variance keeps reducing:



```
3558.50 milliseconds, Variance = 4.14
3553.33 milliseconds, Variance = 2.76
3560.75 milliseconds, Variance = 2.07
3558.80 milliseconds, Variance = 1.65
3558.67 milliseconds, Variance = 1.38
3557.57 milliseconds, Variance = 1.18
3557.62 milliseconds, Variance = 1.03
3557.56 milliseconds, Variance = 0.92
3557.00 milliseconds, Variance = 0.83
3557.09 milliseconds, Variance = 0.75
3556.58 milliseconds, Variance = 0.69
3556.15 milliseconds, Variance = 0.64
3556.36 milliseconds, Variance = 0.59
3556.07 milliseconds, Variance = 0.55
3556.12 milliseconds, Variance = 0.52
```

3. Calibration:

The obtained filtered time values after Kalman filter need to be mapped to a physical distance value. To achieve this, we need to obtain a calibration function for the sensor being used. This function can be obtained as follows:

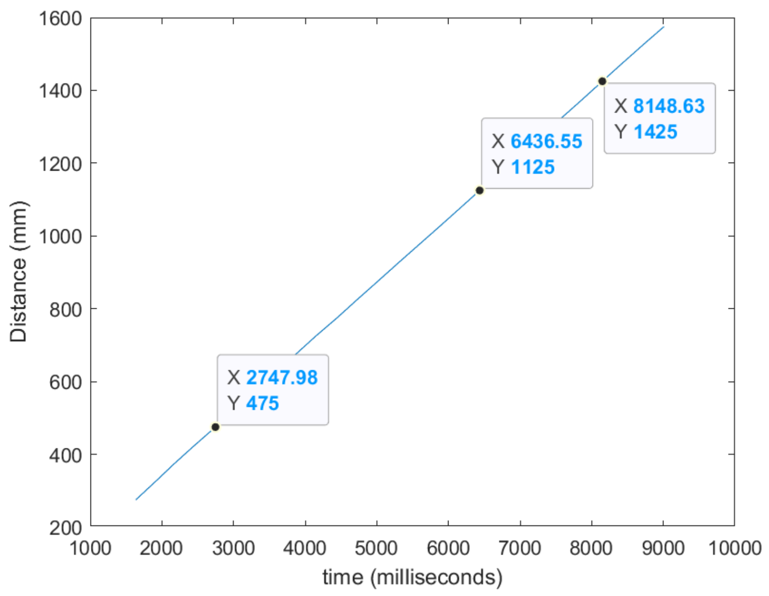
i) Obtain filtered time values for various distances:

The sensor was placed at various distances from an obstacle in front at the **CU-ICAR auditorium** to obtain the closest environment to the testing area. The corresponding filtered time values were noted. Similarly, the actual physical distance was measured using a measuring tape. These data were noted down as follows:

Time (ms)	Distance (cm)
1635.64	27.5
1908.81	32.5
2176.46	37.5
2457.95	42.5
2747.98	47.5
3030.13	52.5
3314.83	57.5
3592.09	62.5
3869.11	67.5
4149.06	72.5
4445.21	77.5
4725.62	82.5
5011.8	87.5

ii) Plot obtained data and Polyfit

The obtained data was plotted in Matlab and polyfit function. Furthermore, since the graph appeared to have regions, we used a piece-wise function for calibration, which was obtained by using Polyfit function on the individual pieces of the graph:



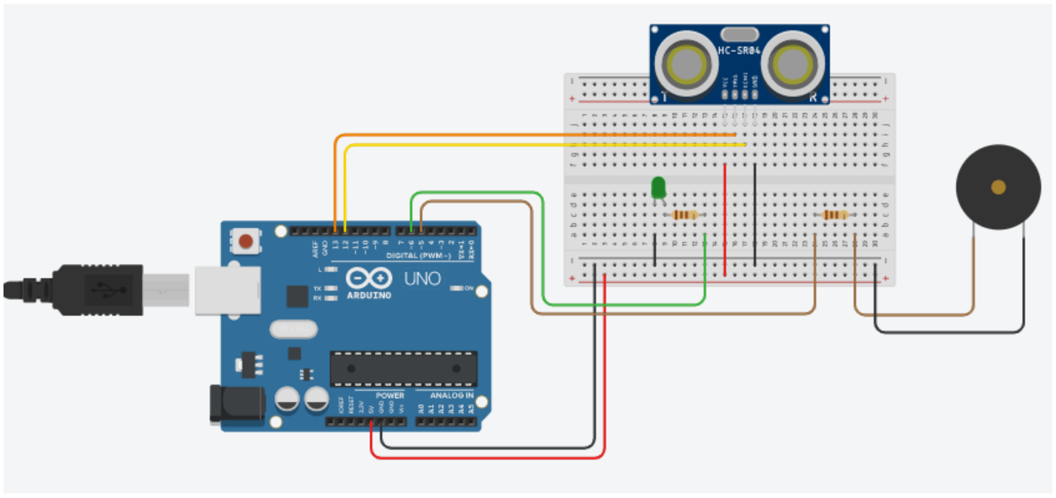
iii) Include Calibration Function in code

The obtained piece-wise calibration function was added to the code to be able to obtain filtered distance values using the sensor.

```
float calibrationFunc(float duration) {
  if (duration <= 2700){
    return (2.244685449327258e-08 * pow(duration,3)) + (-1.507721356370110e-04 * pow(duration, 2))
    + (0.510928188145634 * duration) -2.449521672470731e+02 - 2.75;
  }
  else if(duration <= 6400){
    return (0.175282712005164 * duration) + 2.224399474617530;
  }
  else if(duration <= 8200){
    return (0.174538970666527 * duration) + 8.041191580161483;
  }
  else {
    return (-8.608332581916168e-06 * pow(duration, 2)) + (0.317297053450651 * duration)
    -5.835719432987229e+02;
  }
}
```

1.3 Hardware and Software Implementation:

The following circuit diagram demonstrates the setup used by our group for the project:

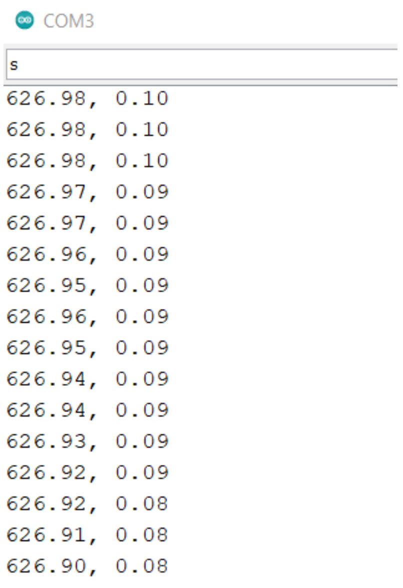


The components used are as follows:

- Arduino Uno R3
- One Ultrasonic Sensor (HC SR04)
- USB Cable
- 1 Breadboard
- 1 LED
- 1 Buzzer
- 2 Resistances (110 Ohms)
- Connecting wires

1.4 Experimental Results

The obtained distance values (in mm) and corresponding error covariance after the Kalman filter can be seen as follows:



```
626.98, 0.10
626.98, 0.10
626.98, 0.10
626.97, 0.09
626.97, 0.09
626.96, 0.09
626.95, 0.09
626.96, 0.09
626.95, 0.09
626.94, 0.09
626.94, 0.09
626.93, 0.09
626.92, 0.09
626.92, 0.08
626.91, 0.08
626.90, 0.08
```

Further Accomplishments:

- The Kalman filter converges within 6 seconds for distance values from 20 cm to 150 cm
- The sensor setup is ideal for measuring distances within the above specified range at the CU-ICAR auditorium.
- Threshold variance for setup = 0.07

Task 2: Sensor Fusion of Multiple Ultrasonic Sensors

2.1 Problem Statement

This task aims to fuse two ultrasonic sensors using the Kalman filter approach with the help of an Arduino microcontroller.

To achieve the above goal, the following steps need to be fulfilled:

- **Sampling**: Sample the time data of at least two ultrasonic sensors as quickly as possible
- **Calibration**: Identify appropriate calibration functions to map the measured time values to the corresponding distance values for each sensor
- **Fusion**: Implement a Kalman filter to fuse the readings from these multiple sensors and estimate the true distance.

Furthermore, the following performance metrics need to be met for the Kalman filter:

- Converge the Kalman filter as quickly as possible (less than 10 seconds)
- Accuracy of calculated distance must be within +/-3 mm

2.2 Technical Approach

The task at hand was fulfilled by completing the steps highlighted in the problem statement as follows:

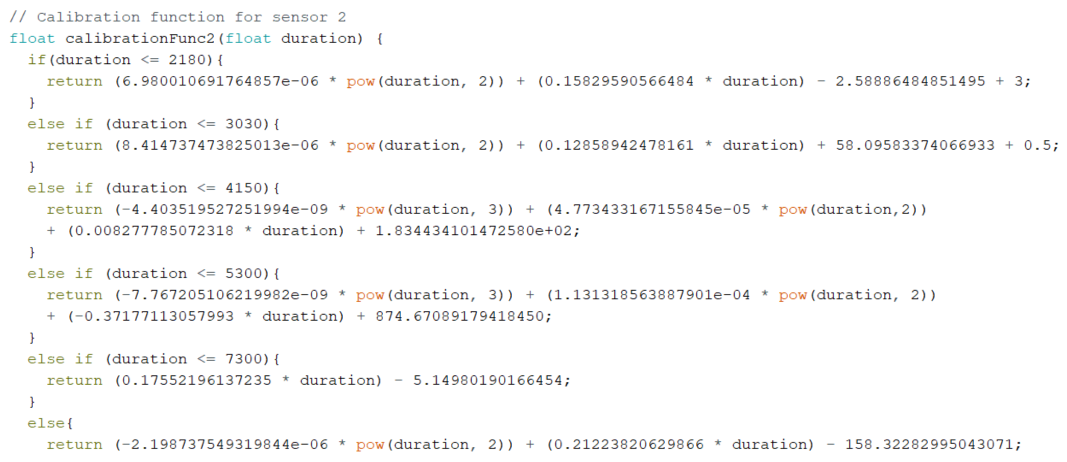
1. **Sampling**

   Time readings were sampled from the two ultrasonic sensors as fast as possible.

   - For our project, a delay of 10 milliseconds was used between measurement readings from the two sensors. Furthermore, another delay of 10 milliseconds was used before taking the next set of measurements from the two sensors.

2. **Calibration:**

   We derive the calibration function for the second sensor in this task (the sensor from Task 1 is being reused for sensor 1). The calibration function for sensor 2 is obtained using the same approach as in Task 1. The obtained piece-wise calibration function is as follows:



```
// Calibration function for sensor 2
float calibrationFunc2(float duration) {
  if(duration <= 2180){
    return (6.980010691764857e-06 * pow(duration, 2)) + (0.15829590566484 * duration) - 2.58886484851495 + 3;
  }
  else if (duration <= 3030){
    return (8.414737473825013e-06 * pow(duration, 2)) + (0.12858942478161 * duration) + 58.09583374066933 + 0.5;
  }
  else if (duration <= 4150){
    return (-4.403519527251994e-09 * pow(duration, 3)) + (4.773433167155845e-05 * pow(duration,2))
    + (0.008277785072318 * duration) + 1.834434101472580e+02;
  }
  else if (duration <= 5300){
    return (-7.767205106219982e-09 * pow(duration, 3)) + (1.131318563887901e-04 * pow(duration, 2))
    + (-0.37177113057993 * duration) + 874.67089179418450;
  }
  else if (duration <= 7300){
    return (0.17552196137235 * duration) - 5.14980190166454;
  }
  else{
    return (-2.198737549319844e-06 * pow(duration, 2)) + (0.21223820629866 * duration) - 158.32282995043071;
```

3. **Kalman Filter**

   As opposed to Task 1, in Task 2 we implement the Kalman filter on distance values (obtained from raw time values). To implement a Kalman filter for two ultrasonic sensors, the following Prediction and Correction equations were used:

   Process Model:

$$y_k = 1*y_{k-1}$$
$$z_{1,k} = 1*y_k + v_{1,k}$$
$$z_{2,k} = 1*y_k + v_{2,k}$$

   where:

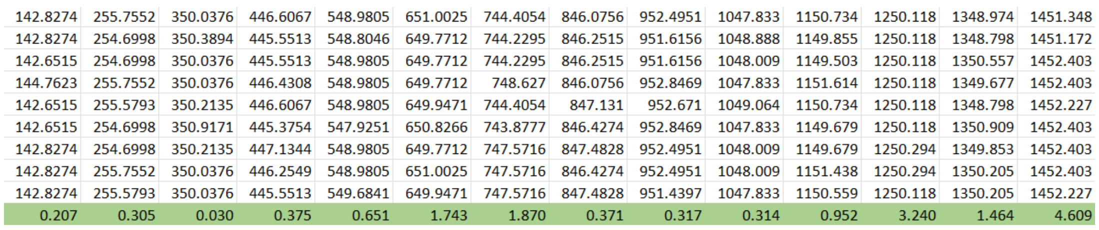
   - y: system state (time values)
   - k: discrete time value
   - z: measured data (time measurements from sensor 1 and 2)
   - v: measurement noise

Prediction	Correction
$\hat{y}^-_k = y_{k-1}$ $P^-_k = P_{k-1}$ where: • $\hat{y}^-_k$: Predicted value based on previous measurement • P^-_k : Predicted error covariance	Correction 1: $K = P^-_k (P^-_k + R_1)^{-1}$ $\hat{y}_{1,k} = \hat{y}^-_k + K(z_{1,k} - \hat{y}^-_k)$ $P_{1,k} = (I - K) P^-_k$ Correction 2: $K = P_{1,k} (P_{1,k} + R_2)^{-1}$ $\hat{y}_k = \hat{y}_{1,k} + K(z_{2,k} - \hat{y}_{1,k})$ $P_k = (I - K) P_{1,k}$ where: • K: Kalman Gain • $\hat{y}_k$: Estimated State • P_k : Estimated error covariance

i) **Error Covariance R**

To implement Kalman filter using above equations, we first need to determine the error covariance R. **The difference for Task 2 is that the R values are determined using distance data instead of time data.**

A snippet of the collected data can be seen below. The columns show distance values (in mm) for distances ranging from 10cm to 150cm in intervals of 10cm each. The last row indicates the R values for the distance interval.



142.8274	255.7552	350.0376	446.6067	548.9805	651.0025	744.4054	846.0756	952.4951	1047.833	1150.734	1250.118	1348.974	1451.348
142.8274	254.6998	350.3894	445.5513	548.8046	649.7712	744.2295	846.2515	951.6156	1048.888	1149.855	1250.118	1348.798	1451.172
142.6515	254.6998	350.0376	445.5513	548.9805	649.7712	744.2295	846.2515	951.6156	1048.009	1149.503	1250.118	1350.557	1452.403
144.7623	255.7552	350.0376	446.4308	548.9805	649.7712	748.627	846.0756	952.8469	1047.833	1151.614	1250.118	1349.677	1452.403
142.6515	255.5793	350.2135	446.6067	548.9805	649.9471	744.4054	847.131	952.671	1049.064	1150.734	1250.118	1348.798	1452.227
142.6515	254.6998	350.9171	445.3754	547.9251	650.8266	743.8777	846.4274	952.8469	1047.833	1149.679	1250.118	1350.909	1452.403
142.8274	254.6998	350.2135	447.1344	548.9805	649.7712	747.5716	847.4828	952.4951	1048.009	1149.679	1250.294	1349.853	1452.403
142.8274	255.7552	350.0376	446.2549	548.9805	651.0025	747.5716	846.4274	952.4951	1048.009	1151.438	1250.294	1350.205	1452.403
142.8274	255.5793	350.0376	445.5513	549.6841	649.9471	747.5716	847.4828	951.4397	1047.833	1150.559	1250.118	1350.205	1452.227
0.207	0.305	0.030	0.375	0.651	1.743	1.870	0.371	0.317	0.314	0.952	3.240	1.464	4.609

iii) **Code for Kalman Filter**

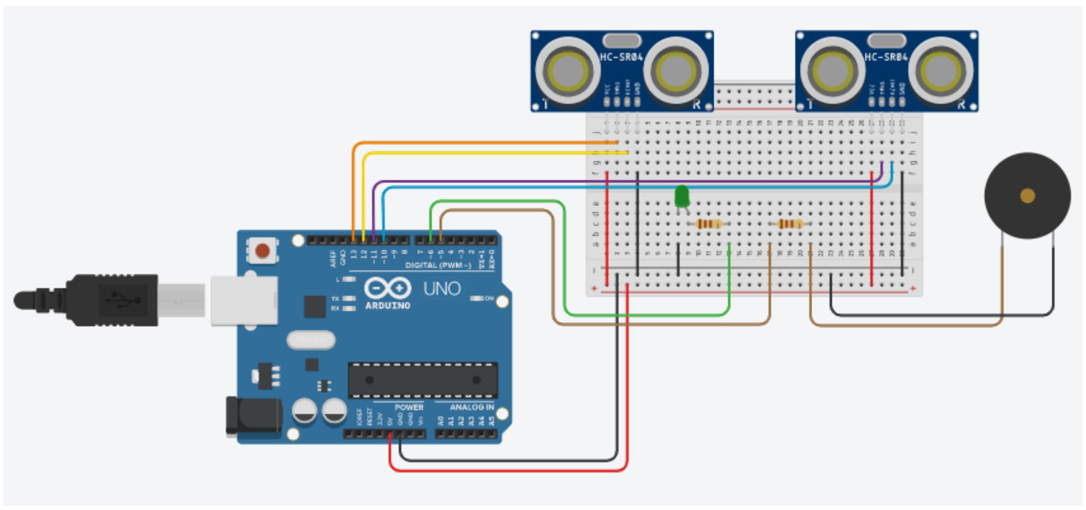
The Kalman filter was implemented on the Arduino using the following code:

```
while(est_variance2 > varThreshold){
  // Prediction Stage
  yPred = yEst2;
  pred_variance = est_variance2;

  // Correction Stage 1
  triggerSensor1();
  yMeas1 = calibrationFunc1(pulseIn(echoPin1, HIGH));  // Distance in mm
  float kalman_gain1 = pred_variance /(pred_variance + measure_variance1);
  yEst1 = yPred + kalman_gain1*(yMeas1 - yPred);
  est_variance1 =  (1 - kalman_gain1)*pred_variance;
  delay(10);

  // Correction Stage 2
  triggerSensor2();
  yMeas2 = calibrationFunc2(pulseIn(echoPin2, HIGH));  // Distance in mm
  float kalman_gain2 = est_variance1 /(est_variance1 + measure_variance2);
  yEst2 = yEst1 + kalman_gain2*(yMeas2 - yEst1);
  est_variance2 =  (1 - kalman_gain2)*est_variance1;
  delay(10);
```

Here, the threshold variance to stop the Kalman filter convergence was taken = 0.0025

2.3 Hardware and Software Implementation:

The following circuit diagram demonstrates the setup used by our group for the project:

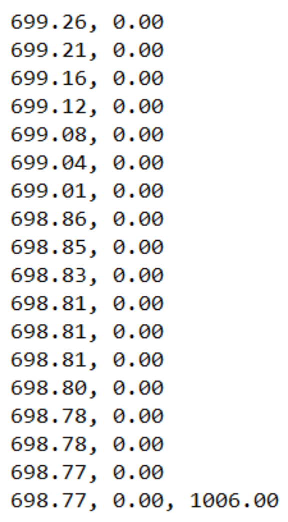


The components used are as follows:

- Arduino Uno R3
- Two Ultrasonic Sensors (HC SR04)
- USB Cable
- 1 Breadboard
- 1 LED
- 1 Buzzer
- 2 Resistances (110 Ohms)
- Connecting wires

2.4 Experimental Results

The obtained distance values (in mm) after fusion and corresponding error covariance after the Kalman filter can be seen in the following snippet. Furthermore, the convergence time of the Kalman filter can also be seen in the last row:



```
699.26, 0.00
699.21, 0.00
699.16, 0.00
699.12, 0.00
699.08, 0.00
699.04, 0.00
699.01, 0.00
698.86, 0.00
698.85, 0.00
698.83, 0.00
698.81, 0.00
698.81, 0.00
698.81, 0.00
698.80, 0.00
698.78, 0.00
698.78, 0.00
698.77, 0.00
698.77, 0.00, 1006.00
```

Further Accomplishments:

- The Kalman filter converges within a maximum of 8 seconds for distance values from 20 cm to 150 cm
- The sensor fusion setup is ideal for measuring distances within the above specified range at the CU-ICAR auditorium.
- Threshold variance for setup = 0.0025

Conclusions and Discussion

3.1 Conclusions

- Kalman filter is a simple yet powerful tool to remove noise from sensor readings. Furthermore, it can successfully fuse data from multiple sensors as well.
- Accurate calibration functions are essential for determining distance values with high precision. While a standard formula using the speed of sound could be used as well, the formula does not capture the variations of the testing environment.
- Fusing two ultrasonic sensors results in distance values with higher precision and reliability.

3.2 Discussion

- Convergence Time:
  - The Kalman filter's convergence time can be substantially reduced by selecting suitable initial values for error covariance

- To determine the suitable error covariance values, we can follow the approach in sections 1.2.2 and 2.2.3 and calculate the R values beforehand. This helps in further reduction of the overall convergence time.

- Sensor Selection:
  - Different sensors have varying degrees of covariances. In order to obtain reliable results, we calculated covariance values for all sensors to determine the most reliable sensors
- Calibration Environment:
  - The data collected to obtain the calibration function, captures the nuances of the surrounding environment. Hence, it is crucial that the calibration environment is kept same as the environment in which the setup will be used.