EEC 10 Final Project Report

Olayiwola David Abraham
March 16th, 2025
EEC 10 Section A05

Table of Contents

Abstract

For our final project, we were instructed to build a sound chasing robot using CCS, a PCB from lab 5, a MSP432, and a TI RLSK Robot. The robot used two microphones to gather audio signals, which were converted from ADC to raw voltage. The voltages were sampled and then filtered using a high and low pass filter, acting as a band pass, to isolate a desired frequency rate of 200 Hz to 3 kHz. This data was then used to find the maximum and average values of both microphones. The average values of the microphone were then analyzed using the code we wrote in CCS studio to find the sound source. Through CCS, we were able to give the robot instructions on when to move forward, left, right, and backward based on the average data from the microphones, as well as the difference between them. We were able to complete most of the objectives for our project, which included stereo recording, audio sampling, filter design, and most motor controls. One area we struggled with in our final project was turning 180 degrees. While implementing our 180-degree turn, we ran into issues with our motors. Whenever our motors would turn for a 180-degree turn, it would have a jerk motion and stop halfway through. Although we did implement the code for our 180-degree turn, we were unable to test the accuracy of the turn due to our motors. Despite this challenge, our robot had consistent and accurate decision-making when it came to hearing and moving towards sound.

Introduction

The purpose of this final project was to test our knowledge on all topics covered in this course. The goal of the final project was to build a robot that could independently find its way to a sound source. The project covered many topics of this course, ranging from operational amplifiers to signal processing, microcontroller programming, motor control, and more.

The first part of this project involved calibrating the microphones of the PCB to ensure a balanced signal reading from both microphones. This step allowed our ADC values to be accurate and ultimately lead to accurate decision-making within our robot. The next part of this lab included audio sampling and filtering of the microphone signals. We did both the sampling and filtering using code written from CCS studio. For our high and low pass filters, we decided to isolate frequencies within 200 Hz and 3 kHz; this allowed the robot to get rid of unnecessary and deterring noise

After the audio signals went through sampling and filtering, we used that data to develop motor algorithms. If there was sound and the right microphone value was higher than the left microphone, then the TI RLSK robot would turn right. It was vice versa for the left side. If there was no sound, then the robot would remain still. With CCS Studio, the robot was programmed to move forward, left, right, and turn 180 degrees. The decisions the robot made were completely based on comparing the averages of the filtered data between the two microphones.

When testing our robot, we came into many obstacles when it came to determining which way our robot should move. While comparing the averages of the filtered data between the two microphones, we used trial and error to find the perfect values that would tell the robot when to move. Our strategy for gaining consistent results included continuous testing and debugging. We used printf statements, watches, breakpoints, and graphs to perfect our code.

<u>Description</u>

The code begins with variable declarations, the main variables being:

- MODE, which is either sampling or running mode. This is used to determine whether we are processing sound or moving. Only one happens at a time in order to prevent the microphones from picking up on the motor noise that occurs during movement.
- DIRECTION, including forward, left, right, and stop. This tells the robot in which direction to go after being assigned a direction.
- There are two Input arrays that take in ADC values from the microphones.
- The ADC values are then converted to voltage values in a Real_Input array
- X and X2 store the raw input data
- Y and Y2 store the high pass filter data
- Z and Z2 store the low pass filter data
- Alpha and alpha_new are filter multiplier values for our desired frequency range
- MaxP6_0 and MaxP6_1 store the maximum values for each set from our Real_Input arrays
- Both avgMax variables store the average of the Max variables, giving us optimal values to work with
- soundLevel is an average of both mics to represent the overall sound level, to check if we have credible sound at all.
- signalDifference represents the difference between the mic voltages to compare the two
- The threshold is the minimum voltage that we consider to be sound due to there being a passive sound level in the microphones.
- I act as an index to go through Real_Input

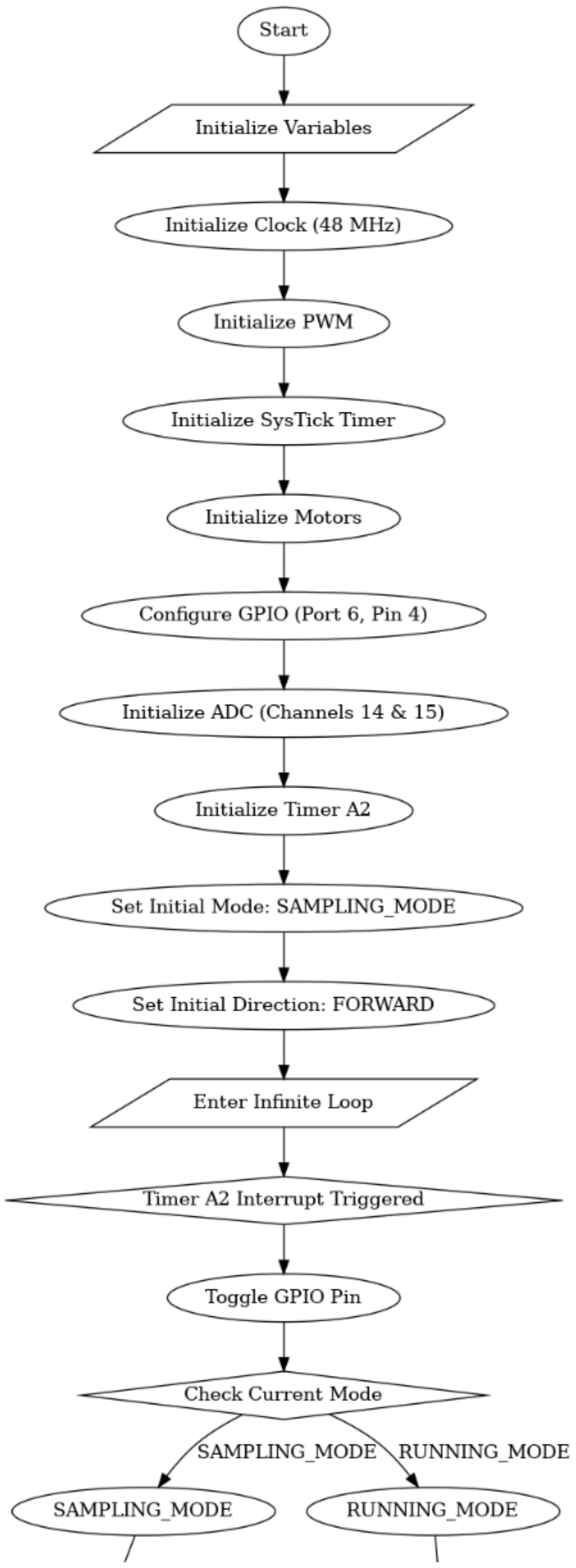
Let’s go through the workflow of the code. First, the robot goes into sampling mode, where it is listening for sound and populating the Real_Input arrays, which is **Part 1,** one for each mic, by converting the raw ADC values into true voltage via a formula. Once the array has been filled, we put the input through a high-pass and a low-pass filter, which acts as a band-pass, ignoring the sounds below and above the cutoff frequencies, which are 200 Hz- 3000 Hz. Anything below 200 Hz or above 3000 Hz will be ignored, and the remaining are transferred into arrays Z and Z2; this is **Part 2.** Once we fill the arrays, we switch MODE into running, then we go through the arrays 100 values at a time, getting a maximum value each time. We take the average of these maximums, and these are avgMax_P6_0 and avgMax_P6_1, corresponding to their respective microphones. We calculate the overall soundLevel by averaging the two individual averages, to know what the overall sound level is. This value is then compared to the threshold, which is set to the passive value that the microphones pick up without any recognizable input. Via this comparison, we determine if there is sound at all. Once we determine that there is sound being played, we compare the two avgMax variables to determine which direction this sound is coming from, assigning that direction to the DIRECTION variable, which are **parts 4 and 5**. If the left side microphone is getting a higher average than the right side, we move left, or else we move right. If the two microphones are getting an approximately equal and recognizable value, we move forward. If there is no sound being played, the robot is told to stop, which is **part 3.**

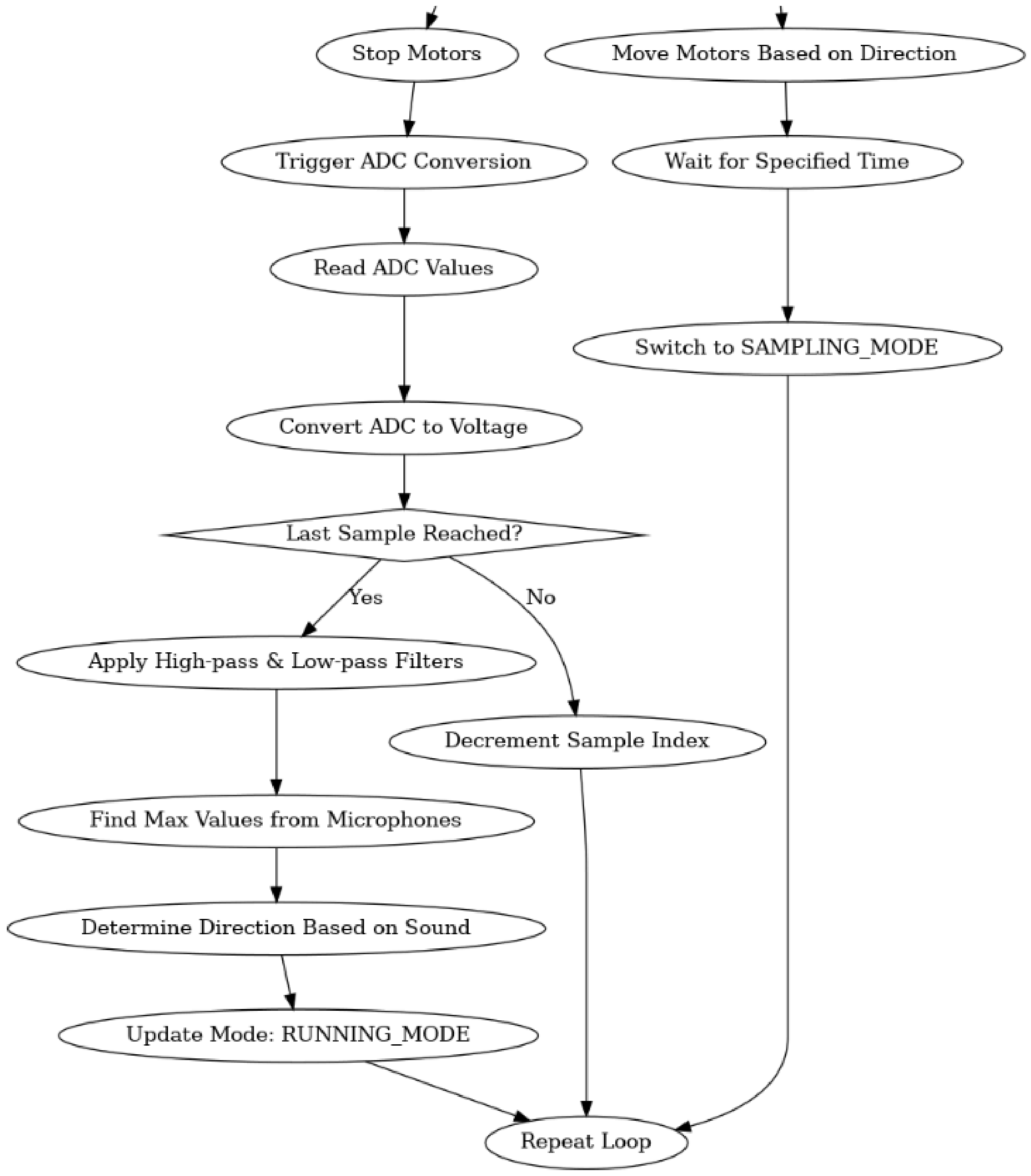
Now in running mode, depending on the direction assigned in sampling mode, we assign a motor function to the robot, either forward, right, or left. To move forward, we move the wheels at the same speed. To move left, we move the right wheel faster than the left wheel, causing the robot to rotate left. Using the same principle for the right, we make the left wheel move faster than the right wheel to turn right. The mode goes back into sampling mode after the motor movement is complete.

During this project, we learned how to construct an inverted op-amp, solder a PCB, use Pulse-Width-Modulation to control motor speed and direction, convert ADC microphone values to voltages, pass voltages through a low-pass and high-pass filter, and analyze and compare the values. We concluded that by using these tools, we can direct a robot toward a sound source.

There is a flowchart included below to show the overall flow of the code.

Flowchart


Start
Initialize Variables
Initialize Clock (48 MHz)
Initialize PWM
Initialize SysTick Timer
Initialize Motors
Configure GPIO (Port 6, Pin 4)
Initialize ADC (Channels 14 & 15)
Initialize Timer A2
Set Initial Mode: SAMPLING_MODE
Set Initial Direction: FORWARD
Enter Infinite Loop
Timer A2 Interrupt Triggered
Toggle GPIO Pin
Check Current Mode
SAMPLING_MODE
RUNNING_MODE
SAMPLING_MODE
RUNNING_MODE



High-Pass Filter Article: https://en.wikipedia.org/wiki/High-pass_filter
Low-Pass Filter Article: https://en.wikipedia.org/wiki/Low-pass_filter

Image of PCB attached to robot with microcontroller