CSE 321 Real-Time and Embedded Operating Systems

Project 3 Report

Prepared By:
Hannah Wilcox

School of Engineering and Applied Sciences
University at Buffalo
December 16, 2021

Table of Contents

Introduction

This report serves to describe the work done on Project 3 for the class CSE 321, Real-Time and Embedded Operating Systems.

This project is meant to act like an automatic trash dispenser, with a purpose of having a hands free way for users to dispose of their garbage. This falls under the autonomous devices design area, and it aids users by not requiring them to have to touch garbage lids or covers when disposing of trash, while also keeping the trash covered so the smell wouldn't be an issue.

Features & Specifications

- Must run "forever"
- When "dumping" certain colored LEDS must light up
- When not "dumping" different colored LEDS must light up
- Servo will move 90 degrees when dumping, then return back to 0 degrees
- User can start servo by waving hand in front of sensor
- Sensor will not take input while servo is running
- User must wave within a specified distance to start servo
- Servo must wait once at 90 degrees for a few seconds before resetting

Internal Integration

For the internal part of this project, the use of multiple techniques were required.

The watchdog timer was integrated in a way to stop the program if the sensor read were to get stuck, as the reading coming in from the ultrasonic sensor is the main piece of the whole project. The watchdog timer is started before the while loop in main, which controls reading in values from the sensor, and gives that loop 2 seconds to kick the timer again before it terminates the program altogether.

Bitwise driver control was used to control the state of two LEDs hooked up to a solderless breadboard. The ports and pins for the LEDs were enabled as outputs, and changed states from being on or off based on the state of the servo motor.

Threads were used to continuously update the state of the servo, one thread being used to run a function with an infinite while loop to check if the servo needed to be moved down and the other thread being used to run a function with an infinite loop to check if the servo needed to be returned to its default state. This allowed for the servo to be

given proper time to move and be checked outside of the main loop, as it would interfere with the time sensitive checking of the ultrasonic sensor reads.

Both threads would need to access and change the value of a flag which is used to check which state the servo needed to be in, running or not. Critical protection techniques were used by using mutexes to prevent the other thread from making a change before the other thread was done with its task.

Interrupt was used on the ultrasonic sensor, the trigger of the sensor is configured as a DigitalOut with pulses being sent periodically while the echo of the sensor is configured as an InterruptIn. Whenever the echo senses an interrupt, a timer starts and then stops when the interrupt is falling. This is used to accurately detect how far away the interrupt was detected from the sensor, and is the main component of this project for detecting if the user is waving their hand close enough to the sensor or not.

Design Process

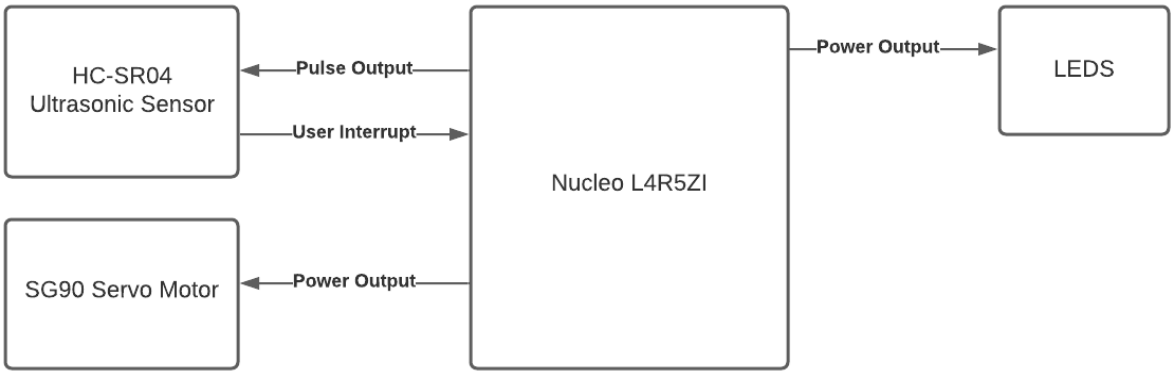
The design process for this project mostly involved how the selected components would interact, and how to have them interact safely.

I knew from the beginning that I wanted to have the components interacting such that whenever some input was felt from the ultrasonic sensor, the servo would rotate to and from a certain position. Once I got the sensor and motor to read values in and move, it was a matter of tweaking how close of a read the sensor should do something for and how long the servo should move for. I decided on the sensor only alerting the servo to move when it got a read of 10 cm or less, and for the servo to move 90 degrees and back.

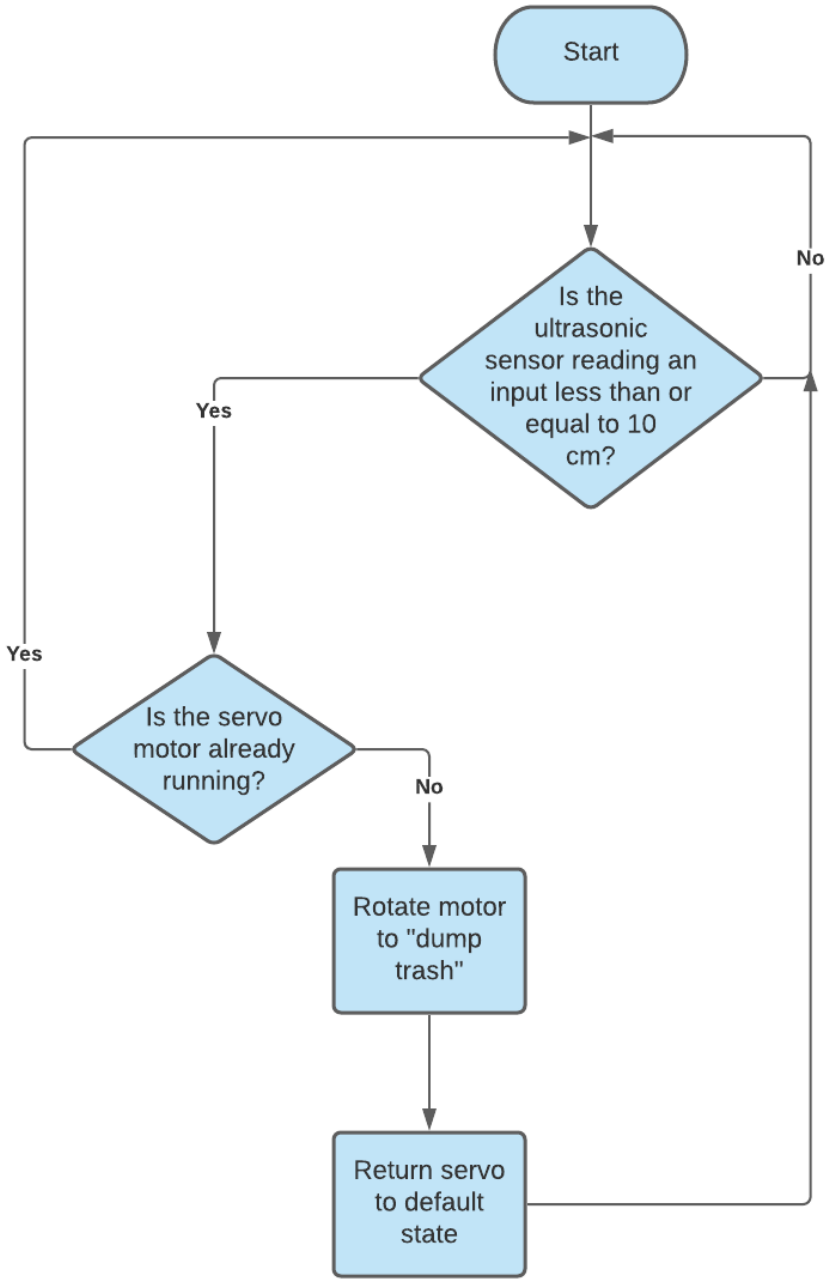
For the internal piece, I knew I had to synchronize the servo and sensor in a way so that when the servo was moving it wouldn't interfere with the time-sensitive process of the ultrasonic sensor reading. For this reason, I decided on using threads to control the servo in the background, while the sensor would be read in the main function while loop. At the same time, I also implemented a watchdog timer in the main function that would restart the program in case the ultrasonic sensor took too long to read or got stuck somehow since the entire project depends on the sensor properly reading in distance values.

In the end, I was able to get the components properly communicating with each other while also implementing all the required internal pieces.

Block Diagram



State Diagram



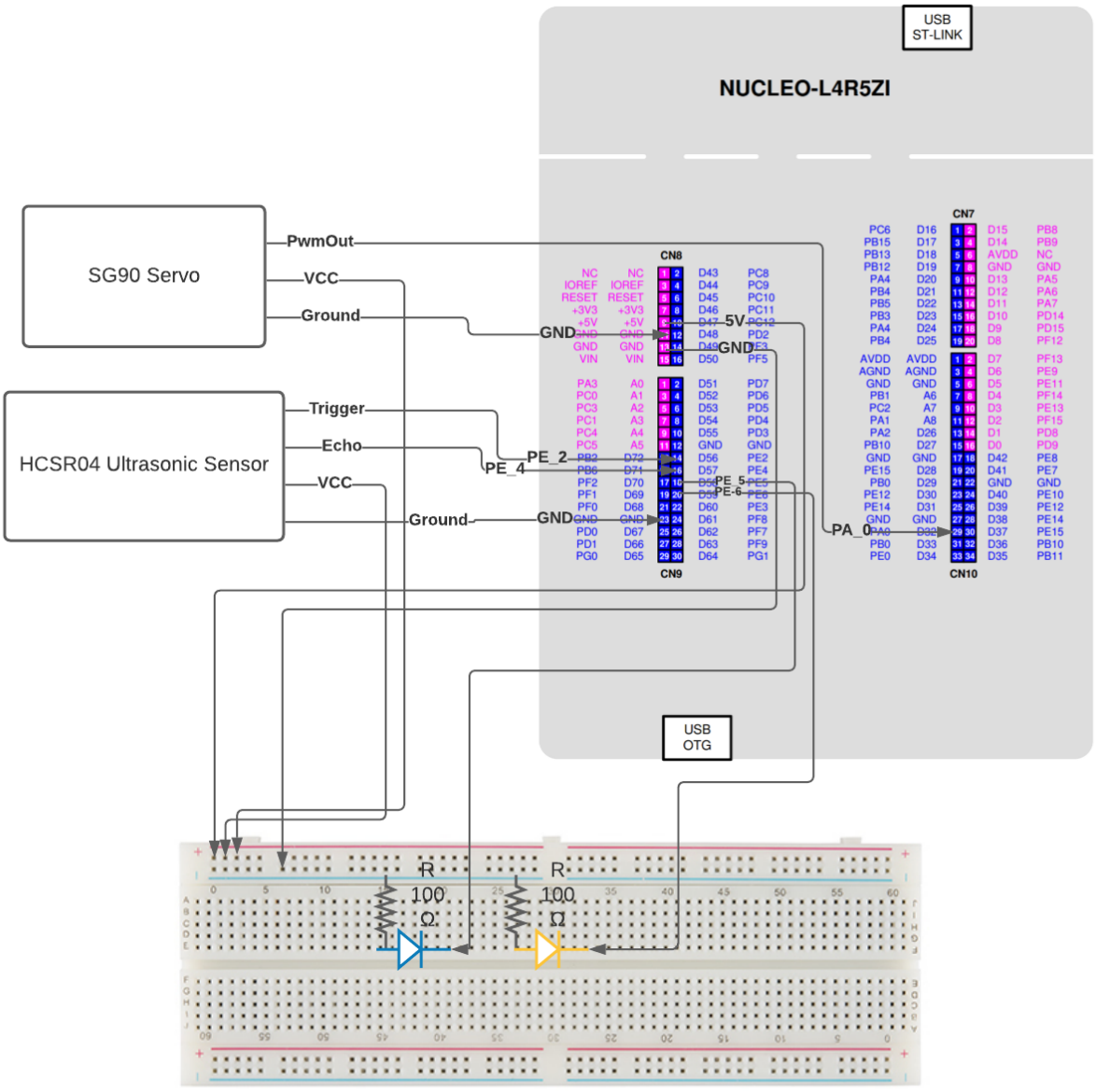
BOM

- LEDS
- Jumper Wires
- HC-SR04 Ultrasonic Sensor
- SG 90 Servo Motor
- Nucleo L4R5ZI
- Solderless Breadboard

Users Instructions

This section will go over how users can build and use given project code. This is assuming that the user has all the required materials listed in the bill of materials section of this report. Included is a schematic diagram which shows users how to wire the different components for the project, how to build the program, and how to use it.

Schematic


USB
ST-LINK
NUCLEO-L4R5ZI
SG90 Servo
PwmOut
VCC
Ground
HCSR04 Ultrasonic Sensor
Trigger
Echo
VCC
Ground
CN8
CN9
CN7
CN10
GND
5V
PE_2
PE_4
PE_5
PE_6
PA_0
USB
OTG
R
100
Ω
R
100
Ω

How to Build

In order to build this project, the user must first configure their components as shown in the schematic on the previous page. Once this is set up, it’s a matter of downloading the code provided on the project’s GitHub and running it on mbed with the machine connected to the users nucleo. Upon building and running the provided code, the user may have to wait up to a minute for the ultrasonic sensor to be fully ready for use, as it takes a moment to take in proper readings. Once this is done the project is ready for use.

How to Use

Using this project is as simple as waving one's hand. All the user has to do once the project is running is wave their hand within 10 centimeters to the connected ultrasonic sensor, and watch the servo move. The servo cannot be interrupted while running, so the user must wait between movements to activate the servo again.

Test Plan Instructions

Testing the proper functionality of this project involved the use of both intrusive testing in the form of print statements, and hardware testing in the form of checking if LEDs were lit up or not.

To test the correctness of the ultrasonic sensor reads, there is a print statement within the code that tells the distance measured in centimeters. This can be used with the sensor at the end of a ruler and using paper in front of the paper. One can check if the distance the paper is from the rule roughly matches that being printed in the console when the code is running.

The servo functionality can be tested by hooking PA_0 (the pin being used on the Nucleo for the servo motor) to an LED. If the code is running properly, the LED should change in brightness. This is useful if one thinks their servo motors aren’t working properly, as it shows that the PwmOut pin on the Nucleo is still functioning. One could also run the code provided in the sources (first link, tagged as the code example provided by a TA) in a separate project in mbed to see if that makes the servo move or not. If that code doesn’t make the servo move, then one can assume they have bad servos and should try with a different servo.

Once these two parts are working, the rest of the project falls into place and should run properly.

Development Timeline

The first stage of development for this project included planning out what peripherals to use, and the basics of how they would interact. In this stage, the stage 1 report was written, and it was decided that this project would use a servo motor, LEDs, and an ultrasonic sensor. The general idea was formed that waving a hand in front of the sensor would activate the servo, but details of how fast this should happen or how close the hand should be were not decided on yet.

Next, forming basic state diagrams for the behavior of the project were developed. The idea of how threads should be incorporated was made into threads being used to control the state of the motor, but no specifics were mapped out yet. Vague planning for how parts should interact was also done, but real progress came when testing out reading the sensor and moving the motor.

Getting a valid reading from the ultrasonic sensor took some time, a few days were spent looking over documentation and trying to apply the API provided by mbed to use on the sensor but with little luck. Office hours were used with a teaching assistant to find out how to get the sensor to work more manually using DigitalOut and InterruptIn by using interrupts to start and stop a timer to measure out how far away a reading is coming in from. Once this was working, the rest of the project started to come together as a specific range of 10 cm was decided on for moving the servo, and watchdog was incorporated to prevent any errors with reading in from the sensor.

For the servo motor, with help from office hours yet again, the servo motor was correctly configured and moving. A mistake with configuring the PwmOut pin on the Nucleo as an output was corrected, as the pin did not need to be activated in the first place. A specific degree of movement of 90 degrees down and back was decided on, with some time between each movement to make it look more “natural”.

Lastly, threads were made to check and modify the state of the servo motor in the background while the main thread took charge of the sensor reading. The changing of LEDs was implemented, as well as various other features as requested from the project document. A day or two was spent on getting proper communication between the sensor and the servo since some timing issues were solved with proper synchronization techniques as described previously in this report.

In total, the project took roughly 3 weeks of planning, coding, and bug fixing to complete.

Recommendations for Improvement

For better improvement, one idea could be implementing some sort of text display to state “CLEAN” if the servo wasn’t active and “DUMPING” while the servo was moving, to make it more obvious to users which state the servo was in rather than just using the LEDs.

Further improvements could include a “cleaning” mode for the motor, which disables inputs from moving the servo to allow for changing out the trash can or cleaning it in any way without causing the lid to move.