Automated Car Avoidance System

Jacob Reyes & Ricardo Rodriguez

December 3st, 2024

Texas A&M University at McAllen - HECM
Multidisciplinary Engineering Department
ESET 349 - Microcontroller Architecture

Instructor: Rafael Fox

TABLE OF CONTENTS

Introduction:

State of the art autonomous vehicles use expensive sensors, cameras, and artificial intelligence software to function. For our final project we wanted to create an obstacle detection / avoidance system using ultrasonic sensors. Our main objective is to create a simple autonomous vehicle by installing our obstacle avoidance system on a motorized vehicle. Our obstacle detection and avoidance system can be used at a small scale to automate small vehicles in predictable environments.

The driving force of our project is the TI MSP-EXP432P4111, using this microcontroller we will interface with the ultrasonic sensors, as well as the motors of the vehicle. The ultrasonic sensors will function as eyes, the microcontroller is the brain, and the motors are the legs. Using the inputs from the sensors, the microcontroller will decide what motors to power in order to avoid any obstacles in front of the vehicle. Of course, the microcontroller will not act on its own. Using assembly language we will program the microcontroller to respond to the ultrasonic sensor's inputs. Once any of the two ultrasonic sensors detect an object within a distance of 25 cm the vehicle will turn in the opposite direction of the sensor that detected the object. For example, if the right sensor detects an object within 25cm the vehicle will turn left in order to avoid the object. The same goes for the left sensor, if an object is detected the vehicle will turn right. When both sensors are detecting objects the robot will stop all together, however, when no objects are detected the vehicle will continue straight ahead.

Applications for small autonomous vehicles include, delivery, transportation, maintenance, security / surveillance, and even entertainment. Our obstacle avoidance system can be a cost effective solution for converting small vehicles into autonomous robots. For example, in a large warehouse a small autonomous vehicle can be used to transport tools or materials across the warehouse. In the same warehouse another autonomous vehicle can patrol after hours, and report to the warehouse owners or authorities in the case of a break in.

Description of System Design:

Hardware:

The Hardware used in this project consists of 5 distinct components. These components are listed down below:

- TI MSP-EXP432P4111 - Microcontroller
- HC-SR04 Ultrasonic Sensors (x2)
- L298N H-Bridge Motor Driver
- 12-V Battery Pack equipped with an On/Off switch
- T Star DSTR robot

The TI MSP432 is a 32-bit microcontroller produced by Texas Instruments, we will be using the MSP-EXP432P4111 (*Figure 1*) version of this controller for our project. Using the IED Keil uVision v5.0 we will program our microcontroller in assembly language, in order to interface with the ultrasonic sensors and motors. We will be using around 13 pins of the microcontroller for our project. The most important pins are the ones that power the microcontroller, control the motors, and read inputs from ultrasonic sensors. Pins 5.6 & 5.7 are outputs to the motors on the left and right side of the vehicle. Pins 2.4 & 2.5 are used to read the inputs on one ultrasonic sensor, while pins 2.6 & 2.7 will be used for the other ultrasonic sensor. Finally pin 2.3 will be the trigger input for both ultrasonic sensors.

The 12 volt battery pack powers the motor driver, a 5 volt output from the motor driver powers the board. Pins 5.6 & 5.7 are the outputs for the motors on the left and right side of the vehicle. Each ultrasonic sensor requires power, ground, trigger, and echo. The Vcc pins on the ultrasonic sensor are connected to the 3.3 volts output on the board, and the ground pins on the sensor are grounded to the board.

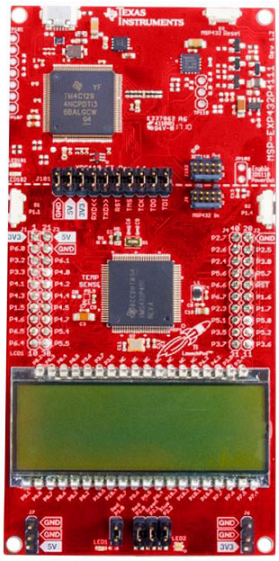


Figure 1: TI MSP-EXP432P4111

The HC-SR04 Ultrasonic Sensor (*Figure 2*) sends an ultrasonic sound from the transmitter and the receiver picks up the signal once it bounces off an object. The sensor then uses internal circuitry to determine the distance between an object and the sensor. The HC-SR04 sensor has four pins, Vcc, Trigger, Echo, and Gnd. The Vcc and Gnd pins are power and ground, the microcontroller will supply 3.3volts to the Vcc pins of both sensors, the Gnd pins of both sensors are connected to the ground pins on the microcontroller. The trigger pin is used to activate the sensor, a 10 µSec pulse is sent from pin 2.3 on the microcontroller which tells the sensor to send out an ultrasonic signal. The Echo pin goes high when the sensor detects the reflected sound wave. The duration that the Echo pin stays high is used to calculate the time it took for the sound pulse to travel to the object and back. In order to calculate the distance an object is we will use the equation: $(Distance(m) = \frac{1}{2} * 343(m/s) * \frac{1}{3MHz} * \alpha)$ where α is the decimal value generated by the sensor. The echo pin for one of the sensors will be connected to pins 2.4 & 2.5, while the echo pin for the other sensor will be connected to pins 2.6 & 2.7 on the microcontroller. The reason why we are using two pins for each sensor is because we are using two capture modes, one timer is capturing the rise of the echo pulse and the other is capturing the fall. Using this method we can

determine the entire pulse width of the echo pin, and plug into our equation to determine the distance between the sensor and an object.

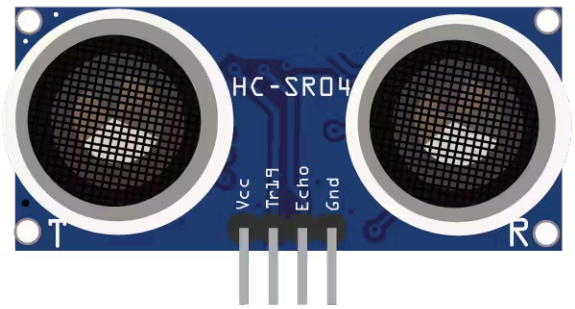


Figure 2: HC-SR04 Ultrasonic Sensor

The L298N H-Bridge Motor Driver (*Figure 3*) allows us to power our motors using the 3.3 volt signals from our microcontroller. The motor driver is connected to the 12 volt power supply from the battery pack, the 5 volt output of the motor driver is what powers our microcontroller. The logic pins on the motor drive is what determines the direction of the motors. Table 1 shows the direction the motors will turn based on the inputs of the logic pins. inputs 1 & 2 are responsible for motor A, while inputs 3 & 4 are responsible for motor B. Currently we have no intentions to make the robot reverse, therefore, we are grounding input pins 2 & 4 of the motor diver to the board. In this configuration the vehicle will only travel forward when input pins 1 & 3 are high. In order to set the speed of the motors we are providing a duty cycle of 10% to 20% from pins 5.6 & 5.7 of the microcontroller to the input pins 1 & 3 of the motor driver. This sets the pace of the vehicle to a reasonable speed that will allow for enough time for the ultrasonic sensors to send and receive signals.

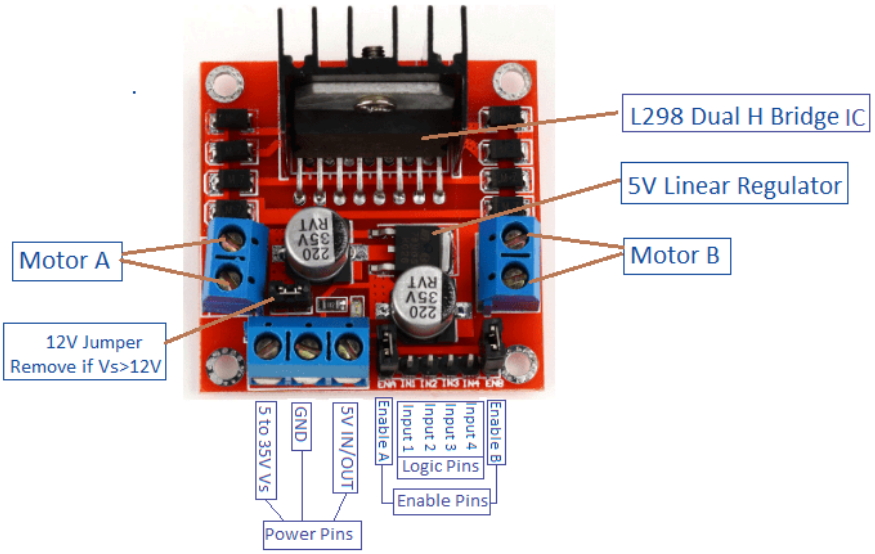


Figure 3: L298N H-Bridge Motor Driver

Table 1: Motor Direction Based on Inputs

Input				Output	
Input 1	Input 2	Input 3	Input 4	Motor A (1&2)	Motor B (3&4)
0	0	0	0	0	0
0	1	0	1	1(Reverse)	1(Reverse)
1	0	1	0	1(Forward)	1(Forward)
1	1	1	1	0	0

The 12-V Battery Pack will power our entire vehicle, as well as provide the voltage needed for our 4 motors. A On/Off switch is wired in series with the power supply and the motor driver, once the switch is in the off position, there will be no power to the microcontroller. We installed this switch to conserve battery when the vehicle is not in use, as well as for the convenience it provides when loading new programs onto the microcontroller.

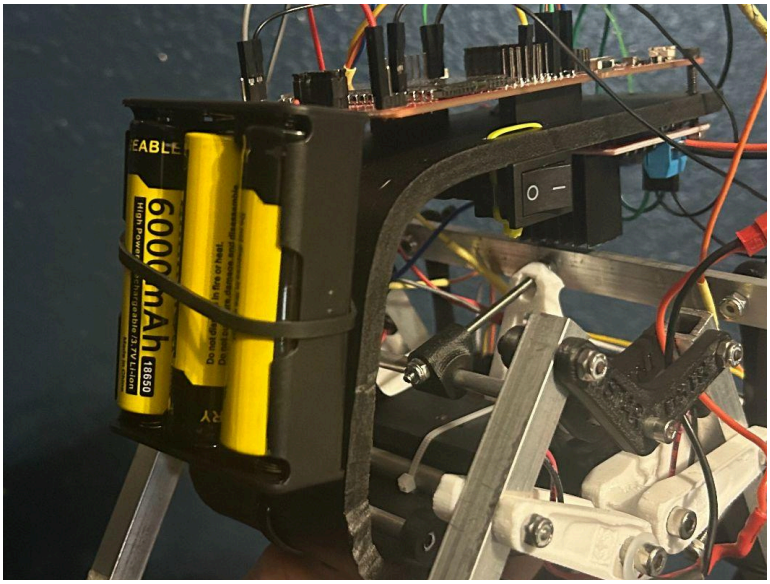


Figure 4: 12-V Battery Pack equipped with an On/Off switch

The T Star DSTR robot will be repurposed for our project, we received the robot in the condition shown in figure 5. Some slight modifications were made to the robot in order to suit our project, these modifications include securing the platform, installing the battery pack, motor driver, and ultrasonic sensors. Due to the increased weight from the battery pack the platform began to tip to one side, we used several zip ties and fastened the platform to the frame of the vehicle. Next, we installed the microcontroller on the top of the platform, and installed the motor driver to the underside of the platform to maximize space. The microcontroller is fastened to the platform using two screws on opposite corners of the board. A combination of screws and zip ties were used to secure the motor driver to the platform. We then mounted an ultrasonic sensor on each of the front supports of the vehicle using foam pieces to separate the sensor from the frame and a rubber band to secure the sensor. When connecting the ultrasonic sensor to the microcontroller, several wires were spliced. The Vcc and Gnd wires from the sensors were spliced together in order to connect both Vcc pins to the 3.3 volt pin and ground pins on the board. The trigger wire on both sensors were spliced so they could be triggered at the same time from the same pin on the microcontroller. Finally each echo pin on both sensors were spliced in order to capture the rising and falling edge of the echo pulse.

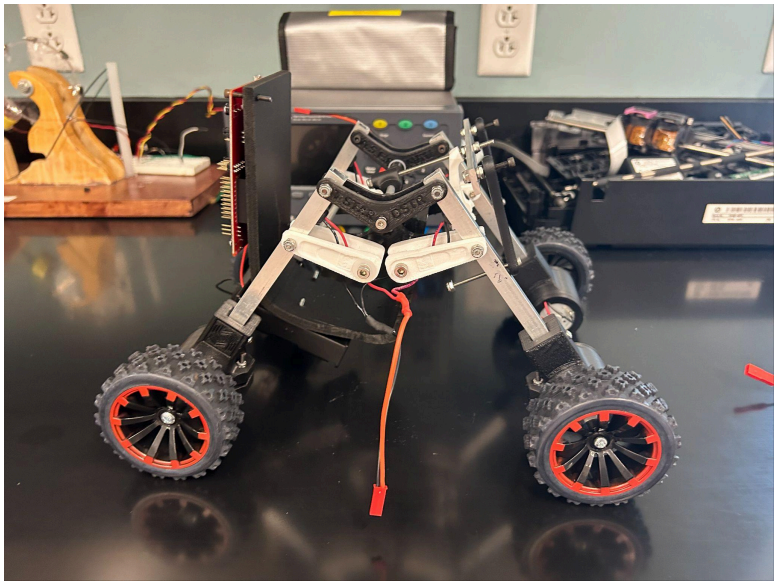
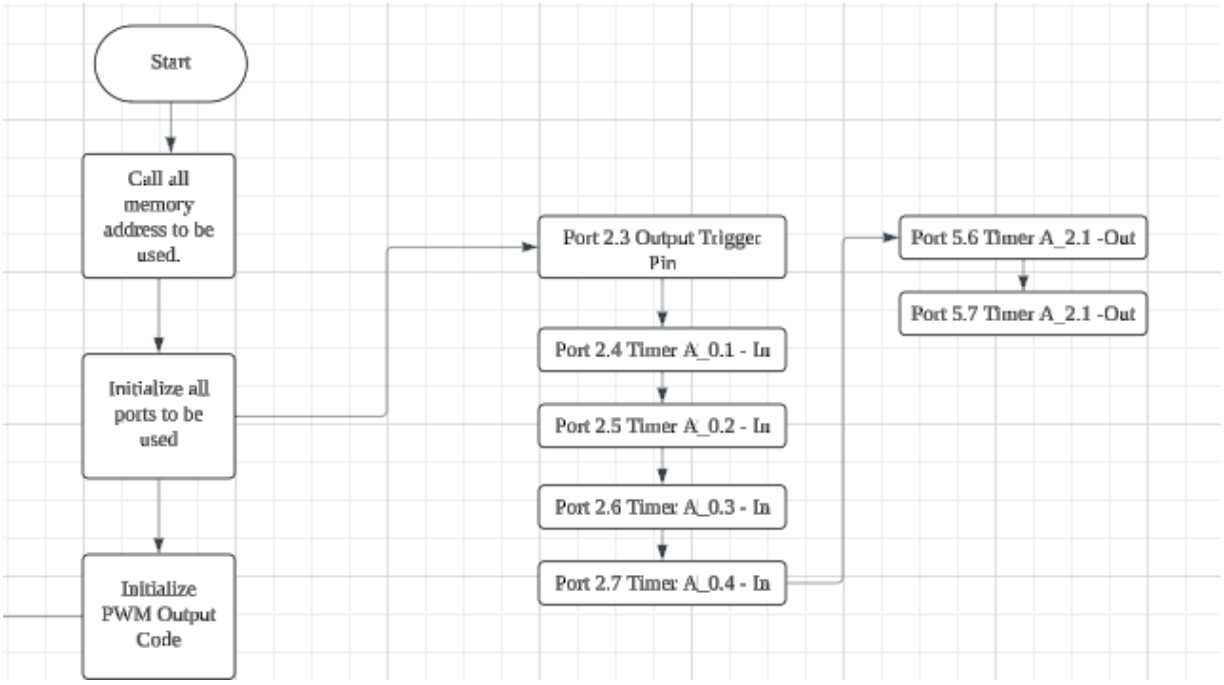


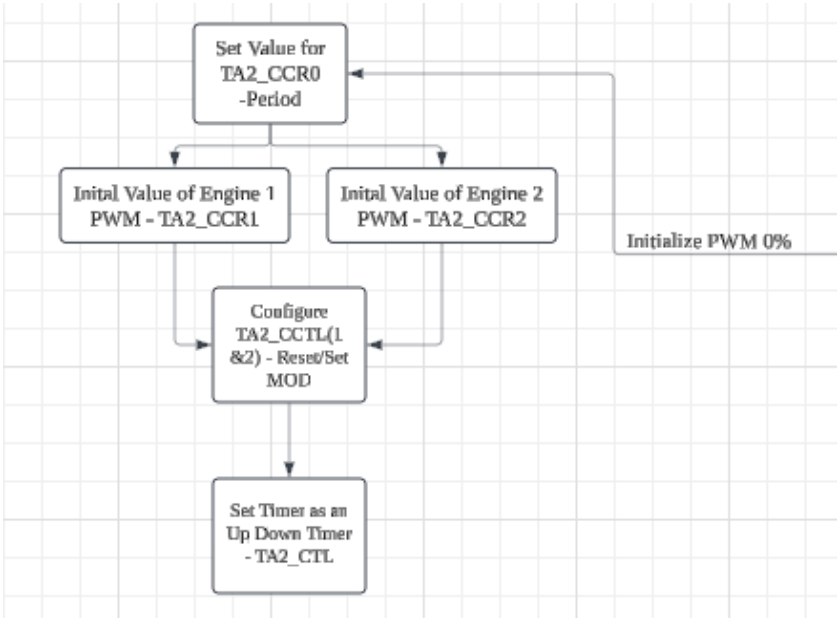
Figure 5: T Star DSTR robot

Software:

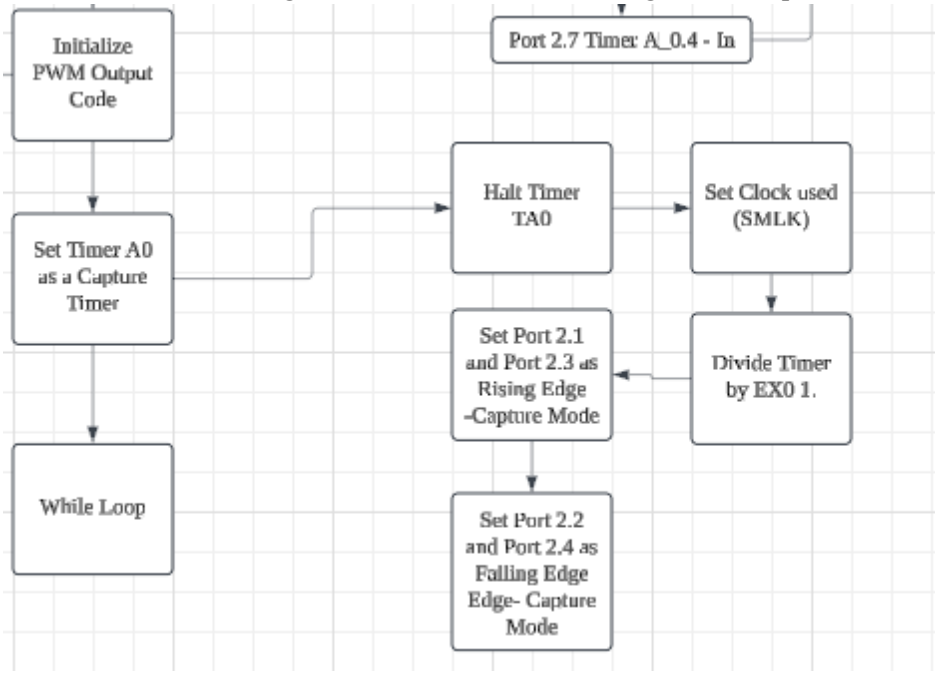
The software we utilized for this Automated Car Avoidance System project would be the IED Keil uVision v5.0 to program our TI MSP-EXP432P4111 microcontroller. This software allowed us to see the registers in the debugging mode, and allowed for quick and easy register manipulation of programing for our project in assembly language. The figure below shows the flow chart we followed to complete the rough programing of this Automated Car Avoidance System.



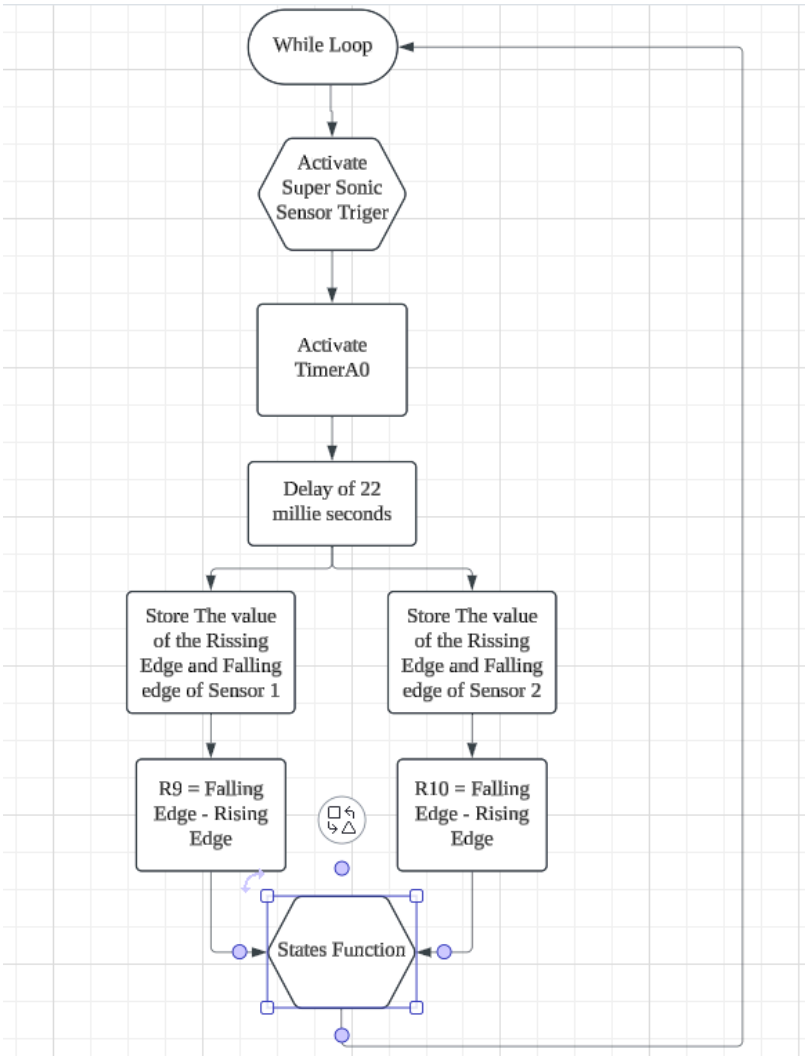
Flow Chart Figure: Initialized Part 1



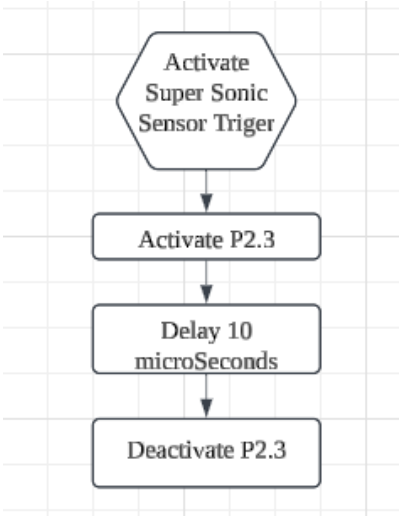
Flow Chart Figure: Initialized Part 2 - Setting PWM Output



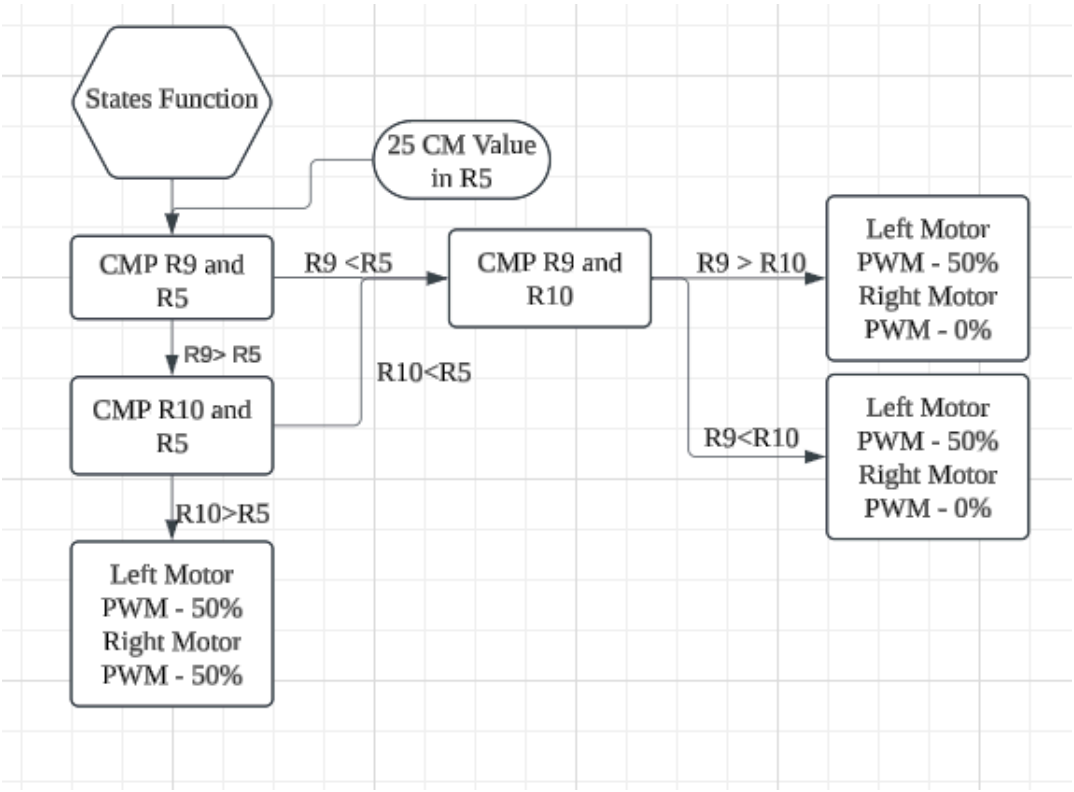
Flow Chart Figure: Initialized Part 3 - Setting PWM Capture Pins



Flow Chart Figure: Part 4 - Setting Main Code Loop



Flow Chart Figure: Part 5 - Setting Function for Activation of Super Sonic Sensor



Flow Chart Figure: Part 6 - Setting Function for States the Robot may Encounter.

As we can see from the various flow charts above, we have various main components to digest for our code examination. Firstly, let us analyse flowchart one. In this flowchart, we can see that we must first define the address of every port, timer, and register we will be using. After declaring a memory address for every port we will be using, then we must proceed to declare in which configuration for the ports we wish to use. For instance, to utilize Timer A_0 we must first select that port configuration by changing the select 0 and select 1 configuration to "01" respectively for Port 2's 2.4, 2.5, 2.6 and 2.7. We must also declare that this Timer A will specifically be an Input, so that we may capture the echo pin generated by the ultrasonic sensor. The code below shows the corresponding code according to "*Flow Chart Figure - Initialization Part 1*".

```
P2SEL0  EQU 0x40004C0B      ;Port 2 Select 0
P2SEL1  EQU 0x40004C0D      ;Port 2 Select 1
P2DIR   EQU 0x40004C05      ;Port 2 Direction
P2_OUT  EQU 0x40004C03      ;Port 2 Output
TA0_CTL EQU 0x40000000      ;Timer A_0 Control
TA0_CCR1 EQU 0x40000014     ;Timer A_0 Read P2.4
TA0_CCTL1 EQU 0x40000004    ;Timer A_0 Control P2.4
TA0_CCR2 EQU 0x40000016     ;Timer A_0 Read P2.5
TA0_CCTL2 EQU 0x40000006    ;Timer A_0 Control P2.5
TA0_CCR3 EQU 0x40000018     ;Timer A_0 Read P2.6
TA0_CCTL3 EQU 0x40000008    ;Timer A_0 Control P2.6
TA0_CCR4 EQU 0x4000001A     ;Timer A_0 Read P2.7
TA0_CCTL4 EQU 0x4000000A    ;Timer A_0 Control P2.7
TA0_EX0   EQU 0x40000020    ;Timer A_0 EX0
P5SEL0  EQU 0x40004C4A      ;Port 5 Select 0
P5SEL1  EQU 0x40004C4C      ;Port 5 Select 1
P5DIR   EQU 0x40004C44      ;Port 5 Direction
TA2_CTL EQU 0x40000800      ;Timer A_2 Control
TA2_CCR0 EQU 0x40000812     ;Timer A_2 Read - Period
TA2_CCTL1 EQU 0x40000804    ;Timer A_2 Control P5.6 - Left PWM
TA2_CCR1 EQU 0x40000814     ;Timer A_2 Read P5.6 - Left PWM
TA2_CCTL2 EQU 0x40000806    ;Timer A_2 Control P5.7 - Right PWM
TA2_CCR2 EQU 0x40000816     ;Timer A_2 Read P5.7 - Right PWM
STACK_TOP EQU 0x20000004    ;Stack Location
; Select 1 and Select 0 Port 5.6 & 5.7 PWM Output Signal
LDR     R0, =P5SEL1
LDRB    R1, [R0]            ;Select 1 = 0
MVN     R2, #0xC0           ; 5.6 & 5.7
AND     R3, R1, R2
STRB    R3, [R0]

;P5 Select_0
LDR     R0, =P5SEL0         ;Select 0 = 1
LDRB    R1, [R0]            ;5.6 & 5.7
ORR     R1, #0xC0
STRB    R1, [R0]

;P5 Direction
LDR     R0, =P5DIR      ;Set Timers at 5.6 and 5.7 as Outputs
LDRB    R1, [R0]
ORR     R1, #0xC0
STRB    R1, [R0]
; Select 1 and Select 0 Port P2.4 & 2.5 & 2.6 & 2.7 PWM INPUT Signal
LDR     R0, =P2SEL1
LDRB    R1, [R0]
MVN     R2, #0xF0
AND     R3, R1, R2      ; P2.4 & 2.5 & 2.6 & 2.7 Pins as Input...
STRB    R3, [R0]        ;PWM signal

;Port 2 Select_0
LDR     R0, =P2SEL0
LDRB    R1, [R0]
ORR     R1, #0xF0       ; P2.4 & 2.5 & 2.6 & 2.7 Pins as Input...
STRB    R1, [R0]        ;PWM signal

;P2 Direction - INPUTS
LDR     R0, =P2DIR
LDRB    R1, [R0]        ;Set 6.6 & 6.7 as a Input
MVN     R2, #0xF0
AND     R3, R1,R2
STRB    R3, [R0]

;Set Port 2.3 as OUTPUTS For Activation Sensor Triger
LDR     R0, =P2DIR
LDRB    R1, [R0]
ORR     R1, #0x08
STRB    R1, [R0]
```

Code 1: Equivalent of FlowChart Part 1

The second section of code we will be analysing would be the activation of our PWM Output using pins 5.6 and 5.7 to ultimately use Timer A_2.1 and Timer A_2.2. To utilize these timers first we must load the value we want our PWM signal to start with. For instance in *Code 2*, we can see how we load the value of 0x0000 into a register TA2_CCR1 and TA2_CRR2 to start the robot at rest. We must not also forget to initialize our PWM signal of 0x EA5F which correlates directly to a 20 mSecond period (50 Hz). We must configure both Timer A_2.1 and Timer A_2.2 as a Reset/ Set mode as well as set our actual Timer A_2 to an Up-Down Timer after setting all the predetermined values. This will allow us to control the speed of the motors using a PWM signal in port 5.6 and port 5.7 by just changing the TA2_CCR1 and TA_CCR2 respectively for each pin.

```
LDR     R11, =0x00000000   ;5.6 Port Initial PWM
LDR     R12, =0x00000000    ;5.7 Port Initial PWM

LDR     R0, =TA2_CCR0
LDR     R1, =0x0000EA5F     ;SETTING THE VALUE FOR THE TOTAL 20MS -50Hz Period
STRH    R1, [R0]
LDRH    R2, [R0]

LDR     R0, =TA2_CCR1       ;SETTING THE INITIAL PWM VALUE FOR LEFT Motor
STRH    R11, [R0]
LDRH    R2, [R0]

LDR     R0, =TA2_CCR2       ;SETTING THE INITIAL PWM VALUE FOR RIGHT Motor
STRH    R12, [R0]
LDRH    R2, [R0]

LDR     R0, =TA2_CCTL1      ;CONFIGURING AS A RESET/SET - Port 5.6
LDR     R1, =0x000000E0
STRH    R1, [R0]
LDRH    R2, [R0]

LDR     R0, =TA2_CCTL2      ;CONFIGURING AS A RESET/SET - Port 5.7
LDR     R1, =0x000000E0
STRH    R1, [R0]
LDRH    R2, [R0]

LDR     R0, =TA2_CTL
LDR     R1, =0x00000214  ; SETTING THE UP-DOWN TIMER - Infinitly
STRH    R1, [R0]
```

Code 2: Equivalent of FlowChart Part 2

The third main component that we must follow would be the initialization of our capture pins for the echo function of the ultrasonic sensor. To do this we must first halt the Timer A_0 and configure it to the correct internal clock (SMLK) and divide it appropriately for our project. After this procedure, we must configure Port 2.4 and 2.6 as Rising Edge Capture mode, which is a value of 0x4900 in a register labeled TA2_CCTL1 and TA2_CCTL3 respectively. Meanwhile, to set Port 2.5 and Port 2.7 as Falling

Edge Capture mode we must load a value of 0x8900 to the register TA2_CCTL2 and TA2_CCTL4 respectively. This concludes the initialization of our capture Timer A PWM input signal and can be seen in figure - *Code 3*.

```
; Port P2.4 & 2.5 & 2.6 & 2.7 as PWM CAPTURE MODE
LDR     R0, =TA0_CTL        ;Halt Timer to Configure
LDR     R1, =0x00000030
MVN     R2, R1
LDRH    R3, [R0]
AND     R4, R3, R2
STRH    R4, [R0]

LDR     R0, =TA0_CTL        ; Set Clock :SMCLK, ID =/1
LDR     R1, =0x00000200
STRH    R1, [R0]


LDR     R0, =TA0_EX0        ; INDEX = /1
LDR     R1, =0x00000007
MVN     R2, R1
LDRH    R3, [R0]
AND     R4, R3, R2
STRH    R4, [R0]


LDR     R0, =TA0_CCTL1      ; Rising Edge , CCI2A, SCS, Capture Mode
LDR     R1, =0x00004900     ;P2.4
STRH    R1, [R0]

LDR     R0, =TA0_CCTL2      ;  Falling Edge, CCI2A, SCS, Capture Mode
LDR     R1, =0x00008900     ;P2.5
STRH    R1, [R0]

LDR     R0, =TA0_CCTL3      ; Rising Edge, CCI2A, SCS, Capture Mode
LDR     R1, =0x00004900     ;P2.6
STRH    R1, [R0]

LDR     R0, =TA0_CCTL4      ; Falling Edge, CCI2A, SCS, Capture Mode
LDR     R1, =0x00008900     ;P2.7
STRH    R1, [R0]
```

Code 3: Equivalent of FlowChart Part 3.

The next crucial section of our code will deal with the continuous loop that will occur for the rest of our program. In this section, we must first begin by having loaded the value of the Timer A_2 Controller Register as well as the activation hex number ready in a register where it will not be distributed. This is done so that when we call the "*Activate_Super_Sonic_Sensor*" subroutine, we will immediately load the activation register to the Timer A_2 Control register after returning from this subroutine. This will allow us to have minimal error with the run time capturing of the echo pin of the sensor. We will have to deploy a 22 second delay to gather the accurate rising and falling edge values and not collect these values prematurely.

The Rising Edge will have to be subtracted from the Falling Edge value to adequately gather the correct length of the PWM signal sent by the echo pin. We will then do this to both Sensor 1, Port 2.5 - 2.4, and Sensor 2, Port 2.7 - 2.6. When we get the subtracted total, we must then store them to register that will be unmoved for the duration of this portion of the code. This is so we may use these registers directly in the next subroutine "*States*", before looping back to the beginning of this program as shown in "*Code 4*" down below.

```
while
        LDR     R9, =TA0_CTL        ;Halt Timer
        LDR     R10, =0x00000024
        LDRH    R11, [R9]

        ; Sub-Routine Function
        BL      Activate_Super_Sonic_Sensor

                                    ;The whole Timer takes 21.845 millie seconds.
        ORR     R12, R10, R11
        STRH    R12,[R9]            ;Activates The CLOCK


        MOV     R0, #22         ; 22 Second Delay
        BL      delayMs

        LDR     R0, =TA0_CCR3       ;Strore The RISING EDGE Capture in R7
        LDRH    R7, [R0]
        LDR     R1, =TA0_CCR4       ;Stores the Falling EDGE CApture in R5
        LDRH    R5, [R1]
        SUB     R10, R5, R7         ;Value of the PWM Length By Subtracting...
                                        ;Falling Edge - Rising Edge

        LDR     R0, =TA0_CCR1       ;Strore The RISING EDGE Capture in R7
        LDRH    R7, [R0]
        LDR     R1, =TA0_CCR2       ;Stores the Falling EDGE CApture in R5
        LDRH    R6, [R1]
        SUB     R9, R6, R7          ;Value of the PWM Length By Subtracting...
                                        ;Falling Edge - Rising Edge

        ; Compare R6 and R7
        LDR     R5, =0x000010EC
        BL      States              ; Subroutine - All States Here

        B   while                   ; Loops Back
Here    B   Here
```

Code 4: Equivalent of FlowChart Part 4.

The final section of the code will deal with the two subroutines mentioned in "*Code 4 & Part 5 - Setting Function for Activation of Super Sonic Sensor*". This subroutine, "*Activate_Super_Sonic_ Sensor*", requires us to activate Port Pin 2.3 high then making a small loop that is approximately 10 micro seconds long to then deactivate the Port Pin 2.3 Output. This small 10µS pulse for the Pin 2.3 is the necessary trigger required to activate the ultrasonic sensor. This code can be described in the following figure "*Code 5*"

Meanwhile, the next subroutine we should examine would be our "*States*" subroutine which chooses how the robot should react when it encounters a specific situation. This subroutine, as seen in "*Code 6* ", will utilize various comparisons. It will firstly compare that both sensor distances are greater than 25 cm, if they accomplish this condition then both the right motor and the left motor will both have a constant and same PWM signal of 20%. Meanwhile, if they fail this condition, then they will enter a second condition that will compare which sensor has returned the smallest distance input signal and then it will result in the opposite change in direction. This will be accomplished by having one motor at approximately 15% PWM and the other motor will result in 0% PWM Signal. This can change due to the sensor orientation.

```
Activate_Super_Sonic_Sensor

        LDR     R0, =P2_OUT
        LDRB    R1, [R0]        ; Activate the Super-sonic Sensor Trigger
        ORR     R1, #0x08       ; P 2.3 OUTPUT - ON
        STRB    R1, [R0]

        MOV     R1, #29        ; do inner loop 29 times
L4_1    SUBS    R1, #1          ; inner loop
        BNE     L4_1            ;Basicaly Wait 10 MicroSeconds

        LDR     R0, =P2_OUT     ;Deactivate the Super-Sonic Sensor Trigger
        LDRB    R1, [R0]
        MVN     R2, #0x08       ; P 2.3 OUTPUT -OFF
        AND     R3, R2, R1
        STRB    R3, [R0]

        BX      LR
```

Code 5: Equivalent of FlowChart Part 5 - Activation Trigger.

```
States
        STR     LR,[R13]
        SUB     R13, R13, #4    ;Stack Pointer Activated
        ;PWM Subroutine uses R11 and R12
        ;Sensor 1 uses R10 and Sensor 2 uses R9
        LDR     R4, =0x000006D5     ; 15CM
        LDR     R5, =0x00000A3F     ; 40CM


        CMP     R10,R5          ;Compare Sensor 1 if grater than 25 CM
        BHS     L1              ; Will Branch if SENSOR 1 is grater than 25 CM
        B       Check
L1      CMP     R9, R5
        BHS     L12             ; Will Branch if SENSOR 2 is grater than 25 CM
        B       Check


Check   CMP     R9, R4
        BHS     Less_Than_5CM
        LDR     R11, =0x00000000    ;PWM SIGNAL 0% - LEFT MOTOR
        LDR     R12, =0x00000000    ;PWM SIGNAL 0% - RIGHT MOTOR
        BL      PWM

Less_Than_5CM
        CMP     R10, R4
        BHS     Less_Than_5CM1
        LDR     R11, =0x00000000    ;PWM SIGNAL 0% - LEFT MOTOR
        LDR     R12, =0x00000000    ;PWM SIGNAL 0% - RIGHT MOTOR
        BL      PWM
        B       enp_loop

ess_Than_5CM1
        CMP     R10, R9
        BHS     LEFT            ; Will Branch if SENSOR 1 is grater than SENSOR 2
Right   LDR     R11, = 0x00000000   ;PWM SIGNAL 0% - LEFT MOTOR
        LDR     R12, = 0x00004650   ;PWM SIGNAL 30% - RIGHT MOTOR
        BL      PWM
        B       Leave_1
```

```
LEFT    CMP     R9, R10
        BHS     Right
        LDR     R11, = 0x00004650   ;PWM SIGNAL 30% - LEFT MOTOR
        LDR     R12, = 0x00000000   ;PWM SIGNAL 0% - RIGHT MOTOR
        BL      PWM
        B   Leave_2


L12     LDR     R11, =0x00004650    ;PWM SIGNAL 30% - LEFT MOTOR
        LDR     R12, =0x00004650    ;PWM SIGNAL 30% - RIGHT MOTOR
        BL      PWM
        B end_loop

enp_loop
Leave_1
Leave_2
end_loop
Less_Than_5CM1
        ADD     R13, R13, #4            ;Loades the stack register info into the link register
        LDR     LR,[R13]                ;Stack Pointer
        BX  LR
```

Code 6: Equivalent of FlowChart Part 6 - States Subroutine.

Test Plan and Results:

The first thing we focused on was completing the assembly of the T Star DSTR robot and determining a safe speed for the robot. The robot we received was not fully completed and lacked a stable platform as well as a DC motor driver. Adjustments were made to the frame of the robot in an attempt to even out the distance of the wheels and secure the platform to the frame. We were able to test a small program to confirm that the motors and the robot in general were working correctly. This program outputs an increasing PWM signal to all 4 motors using a H-Bridge. This PWM signal would increase as we activate press button 1.4 Port in our microcontroller by a 5% increase. We continued experimenting with different duty cycles in order to determine the final speed for our robot. After testing several duty cycles we settled on a duty cycle of 10%. We believe this is a good speed for our experiment, it will give our sensors enough time to gather information and react to obstacles in the robots path.

We then experimented with a single ultrasonic sensor, we conducted a small experiment that consisted of capturing a PWM signal sent by activating our ultrasonic sensor for a 10μs pulse. However, since we were unable to actually see the registers, we determined that we would load the value of the capture into a compare value of 25 cm. If it was greater than this value, then the LED 2 would activate from our microcontroller board. This verified that our ultrasonic sensor was working correctly, the next goal was to interface two sensors with the microcontroller. Once we tried interfacing both sensors we encountered an issue with using the edge capture mode of timer A. The timer was not able to capture both the rising and falling edges of the echo pulse, in order to solve this problem we used two timers for each sensor. One timer would capture the rising edge of the signal while the other timer would capture the falling edge. Using this method we were able to calculate the difference between both captures and calculate the pulse width of the echo signal. This configuration was repeated for the other ultrasonic sensor, in total 4 timers were used to calculate the pulse width of both echo pulses.

We completed the assembly of the vehicle (*Figure 6*) by mounting the ultrasonic sensors to the front of the vehicle and wired the pins to the microcontroller. For our first test we had issues with the sensors responding to obstacles in the vehicle's path. The vehicle would move incredibly slow, due to the fact that only one side of the robot would activate. We addressed this problem by increasing the duty

cycle of the motors to 60% and adjusting the object detection distance required to turn the vehicle. During our final testing phase more issues began to arise, the vehicle was carrying too much momentum causing the robot to crash into objects. More adjustments were made to the software to increase the distance needed to stop the vehicle, which gave the robot enough time to stop.

Finally we were able to test our obstacle detection / avoidance system in the classroom. Once we switched the vehicle on, it began to travel forward at a reasonable speed. We then placed an object in front of the vehicle, which forced the vehicle to stop. We then began experimenting by placing objects in front of one of the sensors, the vehicle responded by attempting to turn away from the object that was being detected. However, the vehicle was not able to produce a steering angle that was great enough to avoid the obstacle. The way we initially intended to turn the vehicle was to power only one side of the vehicle, we believed that the wheels on the opposite side of the vehicle would have enough friction to make the vehicle turn. This was not the case when we performed our final testing of our vehicle, in order for the vehicle to turn away from obstacles we will need to have the motors on one side to go forward, while the motors on the other side go reverse. Unfortunately, due to our time constraints we were unable to implement a reverse option on the vehicle. In the future, if we have the opportunity to continue this project we will implement a reverse function for the vehicle which would allow the vehicle to back away from obstacles as well as improve the steering of the vehicle.

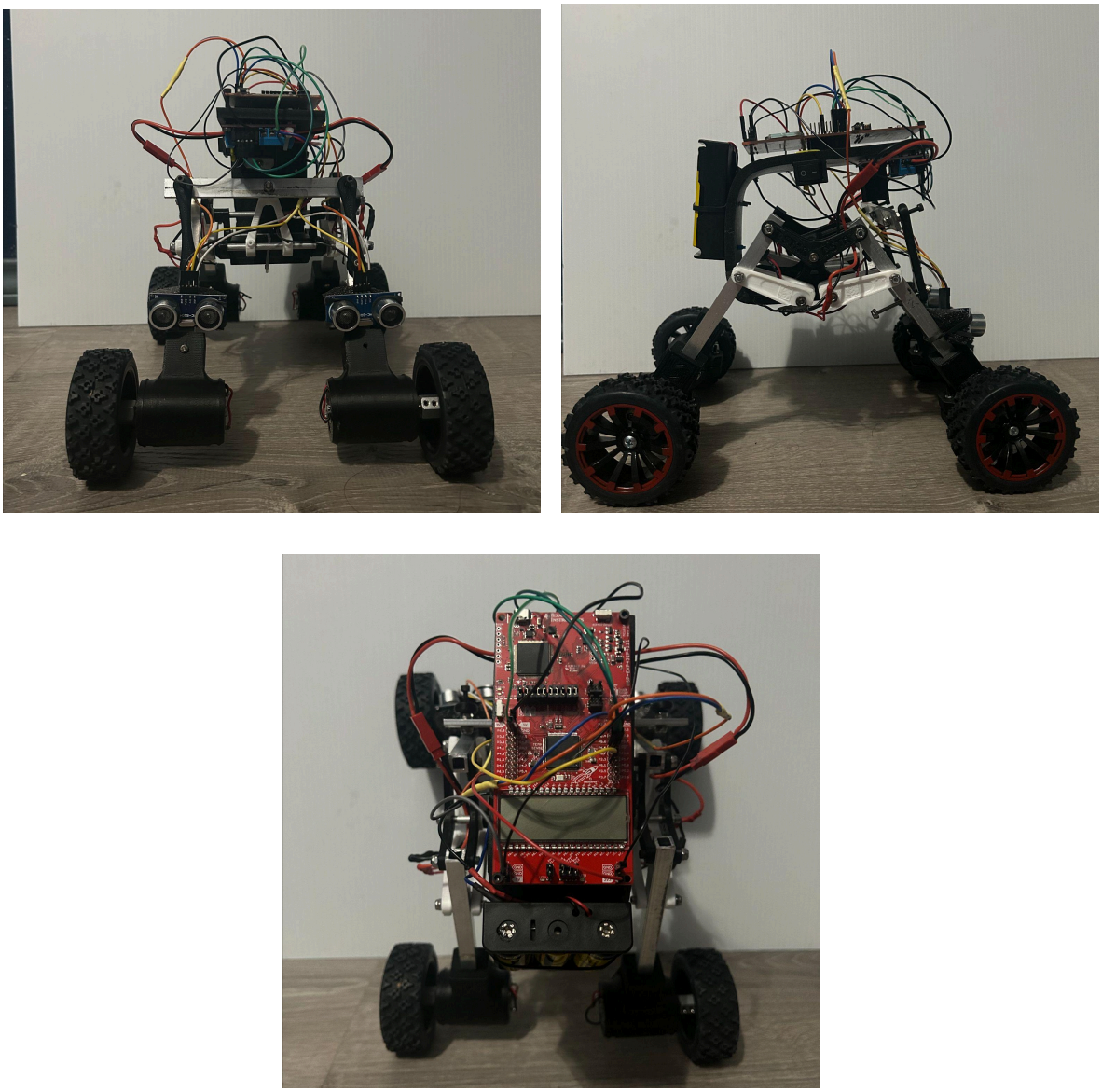


Figure 6: Completed Vehicle, Front View (Left), Side View (Right), and Top View (Bottom)

Description of System Operation:

The Automated Car Avoidance System operates in the following configuration. Once we power our robot, the power from the H-Bridge will rush to power the microcontroller and set in motion our code generated. In theory, the robot will start at rest for the first 25 milliseconds. After initializing *Code 1* through *Code 3,* it will send a 10μS pulse for the trigger of both sensors simultaneously. After the microcontroller decides in which state the robot is currently located, then it will loop again indefinitely.

This Automated Car Avoidance System works best when unhindered in a straight infinite road with the occasional obstacle placed in front of one singular sensor at a time. Although the main basic function of the Automated Car Avoidance System works, we can still strive for more comparisons in our "*States*" subroutine.

Conclusion:

Lastly, we would like to recommend several details that we could have improved upon in this project if the time had allowed for this intervention. The first component change I would like to implement would be the addition of a reverse function for the instance in which the Automated Car Avoidance Car finds itself in front of a wall. . This will allow us to make our project gain one more degree of motion which will be a lot more useful when implementing it in our "*state"* subroutine. Another recommendation we would have wished to implement would be the addition of a feature that would allow for the car to change mode and convert itself into an automated seeking system . These two simple but effective recommendations will allow our project to grow by leaps and bounds in its functionality and diversity.

In conclusion, we learned a lot of various analog components and how to properly connect them to make them work together. A major point/reason to do this project was to challenge ourselves in both creativity and assembly language for the integration and manipulation of Timer A's. This project was not an easy endeavor, a majority of this program was spent researching registers and forms of integrating timers into our code. However, we gained a fundamental footing in assembly language for dealing with Timer A, as well as hardware experience in constructing robots.

References:

This reference deals with all the data sheet registers for every port and Timer A's. This datasheet holds all the information referring to the TI MSP432P4111 Microcontorller.

Citation:

alldatasheet.com. (n.d.). MSP432P4111 Datasheet(PDF). Texas Instruments.

https://www.alldatasheet.com/datasheet-pdf/pdf/1004558/TI1/MSP432P4111.html