SUNY Oswego

Electrical and Computer Engineering Department

ECE492

Hexapod Walking Robot

By

Kara Fortunato, Justin Ross, and Justin Purtell

Supervisors

Rachid Manseur and Marianne Hromalik

Submitted in Partial Fulfillment of the Requirement for the B.Sc. Degree,
Electrical and Computer Engineering Department,

Fall 2018

Project Abstract

Despite large efforts in modern engineering, one fact remains the same when it comes to robotic locomotion: legs and joints will always have a different set of rules than that of a wheeled vehicle when it comes to traversal over flat or rough terrain. A key consideration when designing a "legged" vehicle is how it keeps itself balanced without putting too much weight on one leg and collapsing during motion. In the field of robotics, especially, the ratio between a device's sense of balance and ability to create forward motion in a way that keeps it grounded is highly important in many respects. The Walking Robot project aims to deliver a device that utilizes these concepts to develop a robot that can maintain its own sense of balance while having several degrees of freedom in movement to walk itself in various directions utilizing a series of controllable legs and joints.

Acknowledgment

All gratitude goes to the ECE department and our fellow colleagues for their advice and support throughout the development process. A special thank you to our advisors Dr. Rachid Manseur and Dr. Marianne Hromalik. Furthermore we would also like to acknowledge Dennis Quill for his involvement and advice.

Student Statement

Throughout the research and development process of this project, the designers, Kara Fortunato, Justin Ross, and Justin Purtell, have applied proper ethics to the design process and in selecting the final proposed design. The designers have held the safety of the public to be paramount and have addressed this in the design in all possible aspects.

Table of Contents

Figures & Tables List

Problem Statement

The objectives of this design are to solve complex problems with robotic locomotion by developing a system in which a robot can walk on its own legs and joints while taking into consideration speed, balance, and aesthetic and ergonomic design.

As defined by a set of goals set by the assigning faculty, the design is also to be constructed of a sturdy material with a customized framework that could help accomplish these considerations with more ease than previous attempts at similar designs. This design is also to take into consideration the adjustments necessary to maintain the robots center of gravity always to allow the robot to balance properly during any direction of movement, as well as at any speed within reason. The customized aspect of this design should also give the robot a clean look about it, where the user should not have to worry about any harm coming to themselves or the robot's construction while handling or assembling it.

Finally, the robot should be able to be controlled by some outside system where the user can have full control over various commands such as movement forward and backward, turning left or right, as well as any extra commands that can be programmed into the robot simply to make the robot more appealing to the user. This system could consist of an external controller or application that the user could manipulate in some way to achieve a desired command.

Problem Formulation

The issue of robotic movement is a popular topic amongst modern engineers. When you consider new systems that have been recently attempting to automate the workplace and leisure activities, the importance of robotics nowadays has become more paramount than ever. However, the importance of them operating as intended and being safe to operate are key stress factors in robotic design.

The idea of a walking robot has been established on some level within the electronics market already, therefore is an established concept with its own established drawbacks as well when considering scaling them up to a higher level of use. For example, if a working robot were to be used in the workplace, it would most likely be used to establish transport of materials or

other product from one point of the building to another destination. This considered, the amount of engineering that must go into this type of product is very advanced because, the robot here not only has to manage to balance under the strain of its own weight; its center of gravity and weight distribution is also affected by any product that the robot may be carrying. Therefore, the necessary amount of design consideration changes considerably. On top of this, the engineer must consider the system that will be used to control the program that robot may run on if it is not entirely autonomous. This fine balance between software development and hardware engineering is part of what makes the field of robotics so complex.

In the case of the Walking Robot project, many of these issues were kept in mind throughout the design process due to the increasing need for these kinds of issues to be considered and solved by skilled engineers. For example, the need for it to balance and carry any weight that it must with the amount of hardware mounted to the frame has to be considered for the robot to perform proper movement. If this device were to be used to transport small goods, for example, this consideration would have to be made to account for the weight of anything it would carry as well to keep its center of gravity consistent with its direction of movement.

The control system of the Walking Robot project also has its own considerations towards these issues. The system that drives the robot should be not only simple enough for the user to operate with ease but, it should also allow as much control over the robot's specific program as possible to allow the user to operate it accurately and efficiently in its intended application.

Project Specifications

The design of this project consists of many factors, from aspects such as the framework to programming the system to drive the robot's movement. A strong blend of hardware and software knowledge were implemented to discover an efficient design that would accomplish the intended goals and stay within the constraints specified.

Firstly, the robot needed to be built from custom components that were designed to fit the purpose of the robot itself. The frame of the robot consisted of a series of legs, joints, and a base that could be easily connected and attached together to form the robot. This frame is also designed to house two microcontrollers as well as a series of servo motors that could be used to drive the joints and legs in various directions.

Secondly, a series of programs needed to be developed that could be used to control the servo motors in specific ways such that they could drive the robot in different directions based on how each of the legs and joints coordinated with each other in motion. These programs also take into consideration a method to operate the microcontroller remotely. This program should consist of a series of commands that coordinate with the microcontroller to allow it to activate different operations within the drive program based on user input. These inputs should include, at minimum, a forward drive, backward drive, and rotation left and right, as well as a stop command. These commands should also have the option to run constantly until the user chooses to stop it.

In short, the agreed upon deliverables are listed below:

1. The robot needs to walk forwards, backwards, and turn left and right
2. The robot needs to maintain balance
3. The body of the robot needs to maintain integrity and look semi-professional

Design & Analysis

Number of Legs and Gait

The first thing that needed to be decided was the number of legs that would be used. There are advantages and disadvantages to each number of legs. For example, two legs would require less hardware than a four legged robot, but balancing on two legs is harder than on four. Since balance was the priority, 6 legs were selected. With a basic Tripod gait, at any given time, at least half of the legs are touching the ground, ensuring a balanced center of gravity. Other gaits for hexapods were considered, like the faster bipod gait, but none of them ensured the same balance that a tripod gait does. The minimum number of degrees of freedom required for a hexapod to use the tripod gait is 12 degrees. This directly translates to the number of servos that are required.

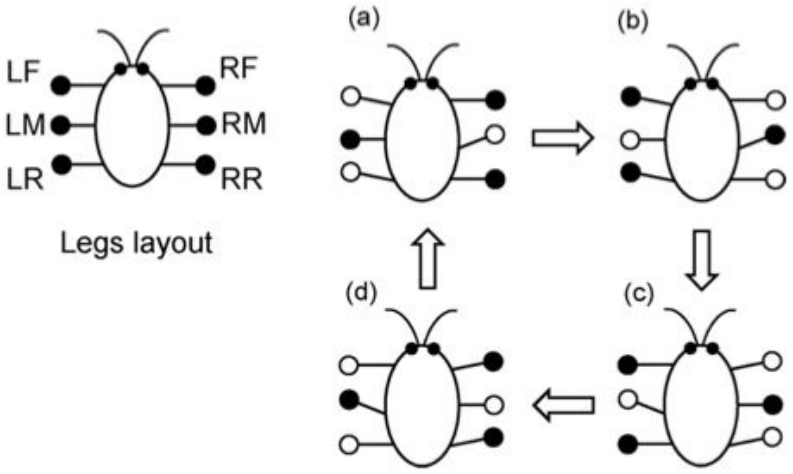


Figure 1: An example of the tripod gait

The forward and backwards follow the Tripod Gait. Which means only three legs move at a time in a triangle shape. The front and back legs of one side along with the middle leg on the opposite side lift up, swing forward, drop down, and finally swing back. When backwards movement is needed, the legs will swing back instead of forward. Turning left and right follows a similar walk cycle. Three legs are on the ground at all times. When turning, the front and back legs swing forward, and the opposite middle leg will lift up and swing back. All three drop down and return back to the neutral standing position. Which way the robot turns is dependent on which sides legs swing forward, and which side swings back.

Texas Instruments TM4C123GH6PM

The microcontroller that was selected to drive the legs on the walking robot was the Texas Instruments TM4C123GH6PM or it was previously called the Tiva-C LaunchPad. The version that was used was still called the Tiva-C LaunchPad, but the new ones are identical. The Tiva-C is based off the 80 MHz ARM Cortex-M4F chipset and has 35 available GPIO pins [2]. It has many available accessories, but the one that was most important to this project was the 16 independent PWM outputs that are generated from two separate PWM modules, each with 4 PWM generators.

Figure 2: The TM4C123GH6PM (Tiva-C) Microcontroller

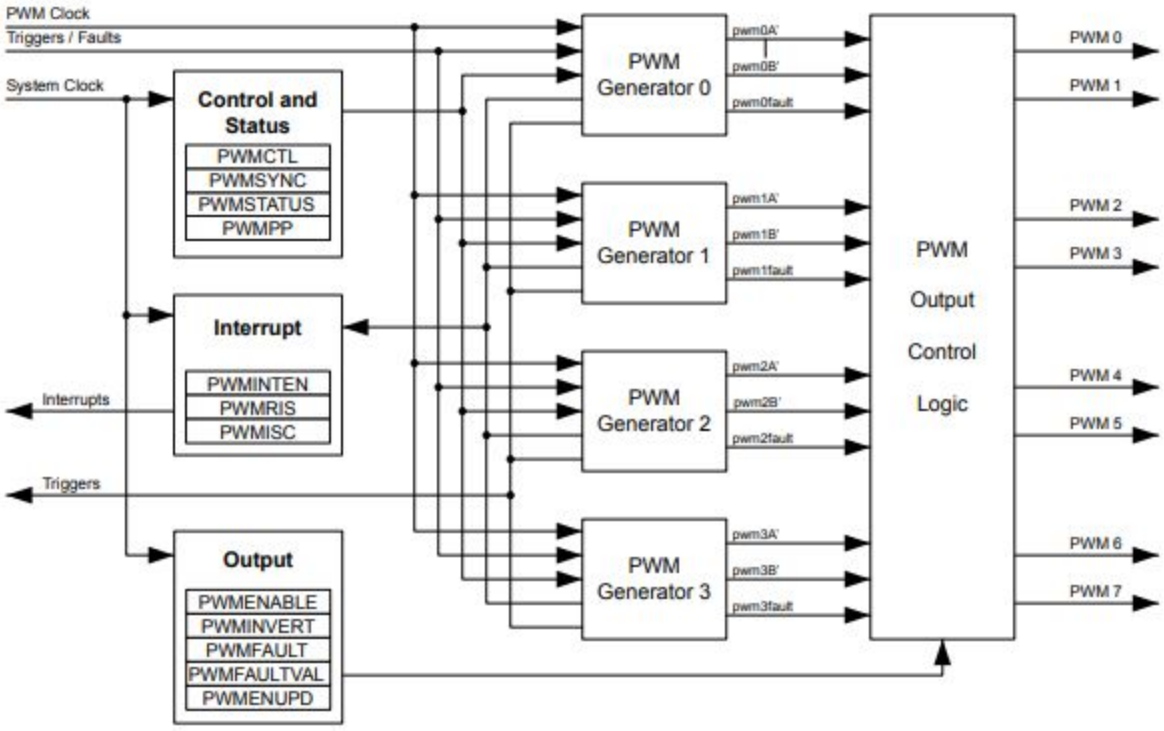


Figure 3: A Diagram of the PWM Generators

The Tiva-C can be programmed using the C99 standard of the C Programming language, and can be written to using a micro-USB-B 2.0 cable. The software used to write and debug code was Code Composer Studio v8 from Texas Instruments. It's a free program that allows the

composition and debugging of code for the Tiva-C, among other Texas Instruments microcontrollers.

Texas Instruments had created a library called “driverlib” that has a variety of register level abstractions that makes programming the Tiva-C a bit easier. Instead of moving individual bits in registers, higher level functions can be called to make the readability of the code easier. But, if necessary, the ability to move individual bits in registers is still available to the user.

The Tiva-C was chosen over other available microcontrollers for two main reasons, versatility and availability. As previously discussed, the Tiva-C has a load of available functions, so the potential for the addition of more features is possible. There was also more than enough PWM pins for the robot. The Tiva-C was also available for free from the ECE department, which helped inform our decision.

Miuzei SG90

It was decided early on that a design goal was to minimize the weight of the walker, so a micro servo form factor was selected. The next thing that was considered was the type of gearing, metal or plastic. The metal gearing is known to be more durable, but they are heavier and more expensive than plastic. Considering this, the Miuzei SG90 micro servos were selected. They have plastic gearing and were lighter and significantly cheaper than the metal counterparts.

Twelve SG90’s were used, and they are controlled with a 55 Hz PWM pulse train from the Texas Instruments TM4C123GH6PM microcontroller. The motors have a 120° range of motion, which was more than enough to ensure movement [3].

Figure 4: Miuzei SG90

Individual Pieces of build

The frame is designed using the 3-dimensional computer aided design (CAD) software AutoDesk Inventor. The components are laser-cut from acrylic sheets to ensure that the design was not only pin-point accurate but also sturdy enough to hold up during the next phases. The use of a 3D printer was considered, but it was determined that laser cutting would be a better approach for this project. The ECE department has access to a Universal laser systems laser cutter, making it easy for students to create and cut components without going through a third party. 3D components also cost money according to how long the project takes and how much material is used. Laser cutting has no cost except for the acrylic, which is provided by the department. Additionally, laser cutting is much faster than 3D printing. The entire project takes less than ten minutes to laser cut while a 3D printer might take hours. The fast cutting time allows for models to be adjusted or completely reprinted completely. This is necessary because the robot was completely redesigned several times.

The first part of the frame to be considered was the shape and construction of the base. For this, an octagonal shape is used to allow the robot to consist of 6 legs, 3 on the left side and 3 on the right side, as well as including ports for all the necessary hardware mounting. Holes were made in the base to hold the servos and so that the microcontroller circuit could be mounted.

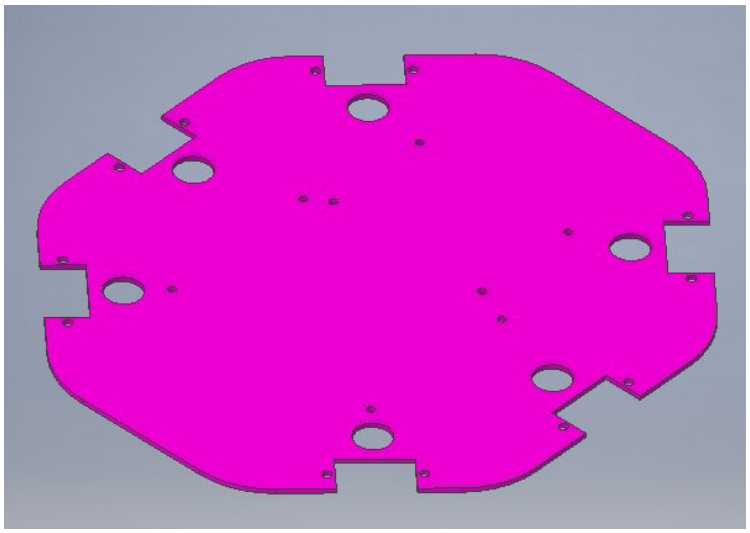


Figure 5: Frame Base

Next, the legs and joints are designed such that they could hold the servo motors in place and use them for movement while remaining locked to the joints and base by mounting screws and acrylic epoxy. The legs are cut with a concave, pointed “toe” that are coated with a rubber tip such that they could have a strong connection with the ground without slipping. The joints each have cut-out slots that connect each of the joint types to one another. A screw-mounted joint is also used on each side of the servos on the legs and on the base to help lock them in place during movement and keep them connected to the frame.

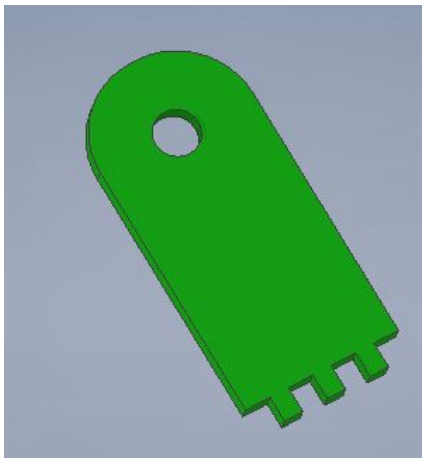


Figure 6: Servo Mount

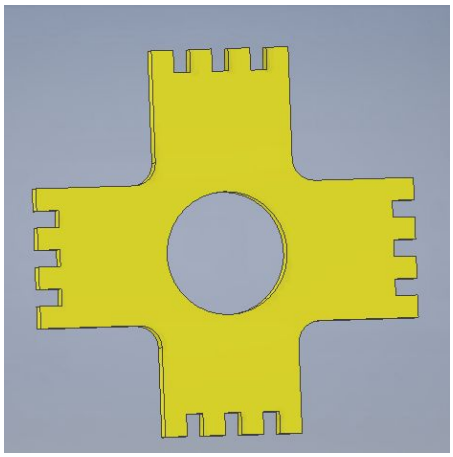


Figure 7: Cross Connection Joint

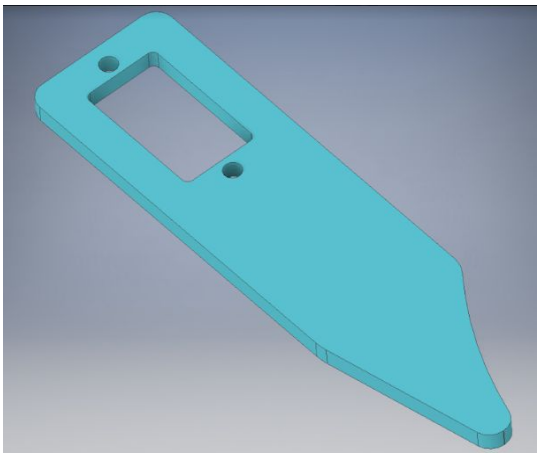


Figure 8: Leg

Once assembled, each leg joint consisted of one cross connection joint, four servo mounts, one leg, and two servos. Each servo was given a number to be referred to. The servo that would be connected to the based was dubbed servo “one” while the servo directly connected to the leg was called servo “two”. This allowed for each servo to be distinguished when the circuit was being built. The final leg joint assembly can be seen in the following figure.

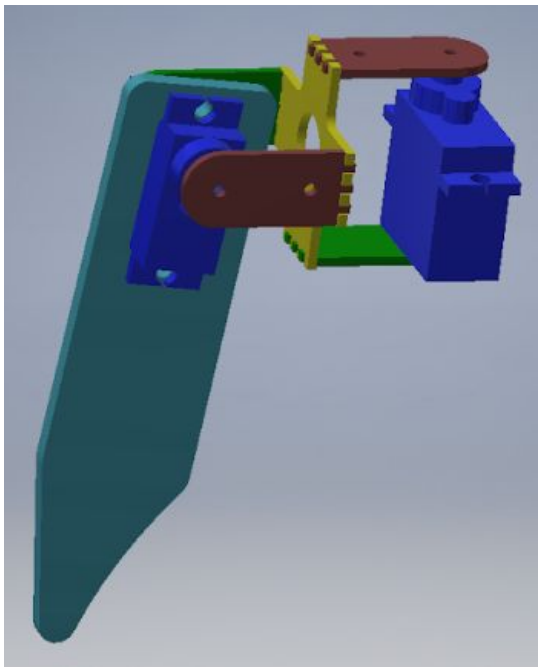


Figure 9: Joint and Leg Assembly

NodeMCU V1

To control the robot, a method of control needs to be selected. Wireless control was chosen because of accessibility. Anyone with a smart device should be able to have access. To accomplish this, the NodeMCU V1 was used. It is a WiFi-enabled microcontroller that uses the ESP8266MOD chip and can broadcast WiFi in accordance to the IEEE 802.11 b/g/n standard [4]. The NodeMCU has the capability to act as a soft access point that can serve web pages and was programmed using a modified version of C++ in the Arduino IDE. The NodeMCU also supports Lua scripting, but there were issues with the serial connection when the firmware is set for Lua scripting. The only way found to ensure a reliable experience was the use of the Arduino IDE. The NodeMCU was chosen over a Bluetooth module because of the accessibility it provides. Any WiFi enabled device can access and use the walking robot, but a Bluetooth module would require either an OS specific app, or the use of a generic Bluetooth command line.

Figure 10: The NodeMCU

To coordinate with the WiFi module, the program allows the user to connect to the module's IP address where a web application is loaded that allows the module to communicate with the microcontroller to activate various methods in the servo program based on user command. The application consists of commands including, forward, backward, rotate left, rotate right, stop, wave, continuous operation, and a speed toggle. The web application was written using HTML and AJAX (Asynchronous JavaScript and XML). These commands interact with the TM4C123GH6PM via a UART bridge at a 115200 baud rate, which cause an interrupt, and the previous command is saved to a global variable. Each possible command is mapped to a specific movement function.

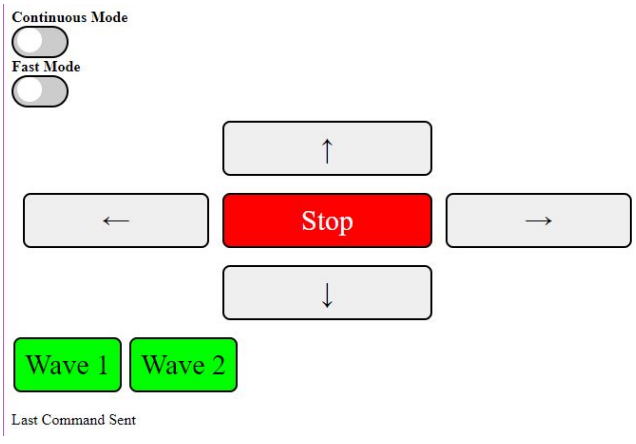


Figure 11: Web Interface

Circuit

To tie all the components together, a circuit needs to be constructed and soldered together. Each of the twelve servos need to be connected to its respective PWM pin on the Tiva C. The UART bridge between the Tiva-C and the NodeMCU needs to be put in place. And each of the components need to be powered by an external benchtop power supply. The power supply's point of entry is the two wires attached to a switch to turn the circuit on or off.

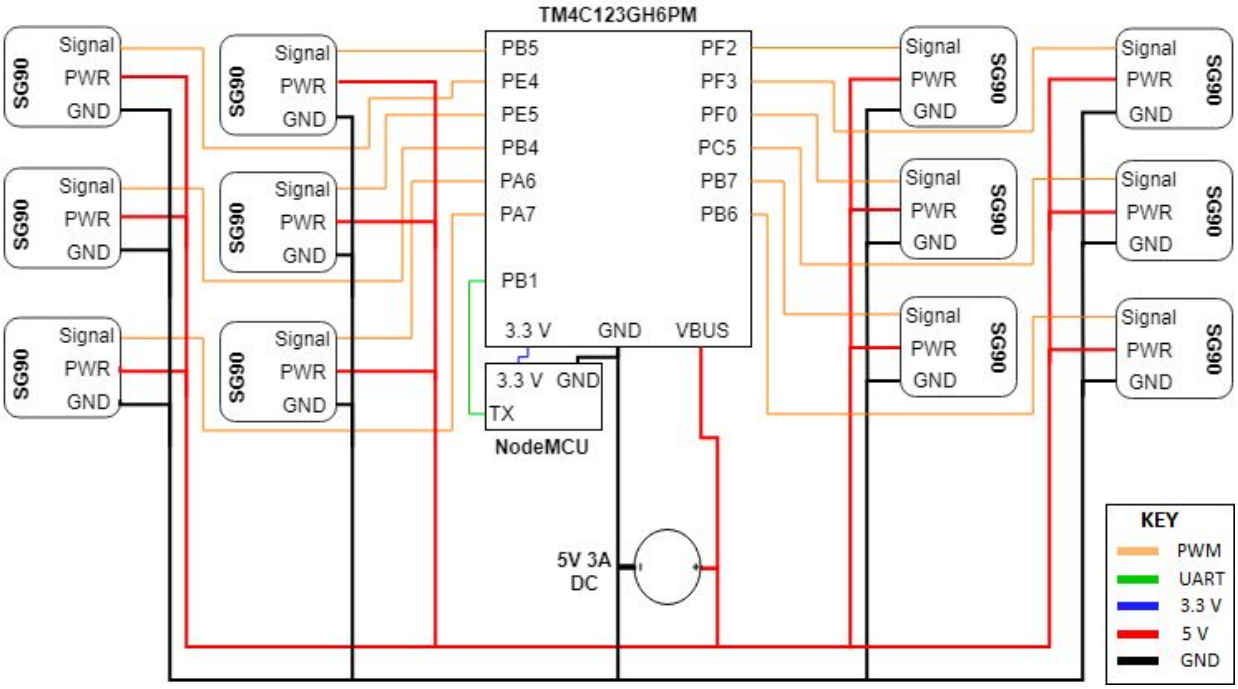


Figure 12: Wiring Block Diagram

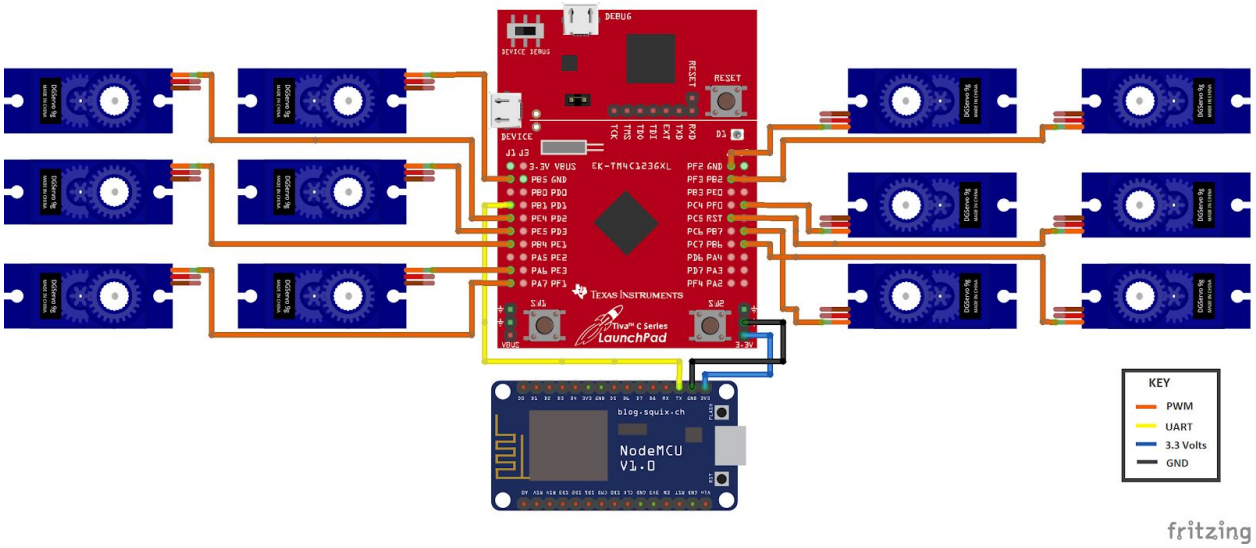


Figure 13: Signal Wiring Diagram

To ensure consistency throughout all aspects of the project, a convention for the legs in terms of the circuit positioning needed to be determined. The base was divided into 3 sections; Front, Middle, and Back. The left side has 6 servos, and the right side has 6 servos. The servos closest to the body are denoted as a servo 1, and the ones on the legs servo 2's. Knowing what side, left or right, and what third of the base the servo is on, the servo number can quickly be determined. The front needed to be distinct to differentiate the left from the right side and the back from the front. The way that the front was denoted was by the row of googly eyes.

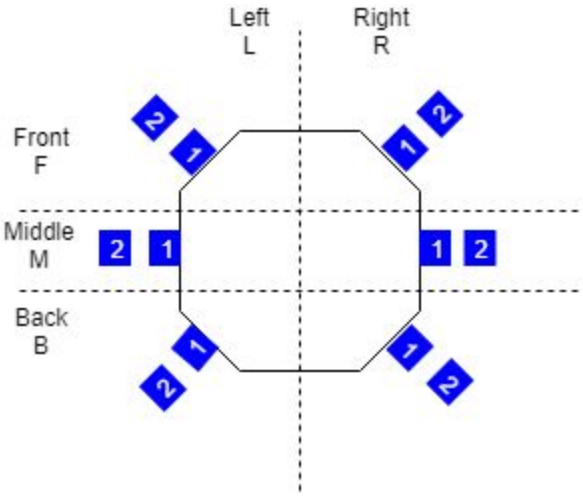


Figure 14: Servo conventions

The physical construction of the circuit was simply done with a 6x8 cm and a 4x6 cm double-sided prototyping PCB. The NodeMCU was placed into soldered female headers on the 4x6 cm board, and the Tiva-C was placed onto soldered male headers on the 6x8cm board. The

connections are as shown in the figures above. There were some obstacles that were faced with the soldering, most of which come from inexperience. The biggest issue come from the nature of a solid core wire. Those wires are much more rigid than a braided wire, and are very easy to break.

The final build consisted of six legs and their corresponding joints, as well as one base. Servo “one” is the servo that is directly connected to the base while servo “two” connects to the leg. The final necessary design to be implemented is the mounted PCB. Mounted to the PCB are the WiFi module, the microcontroller, and a switch that can be used to turn the robot on and off. The PCB is then mounted onto the frame’s base using through-hole standoffs of size M2. These components are soldered to the prototyping PCB, as well as each of the PWM ports and WiFi module ports to allow all of the signals to be properly routed to their destinations. To power the robot, a lab bench power supply was set to 5 Volts and 3 Amps and then attached directly to the switch.

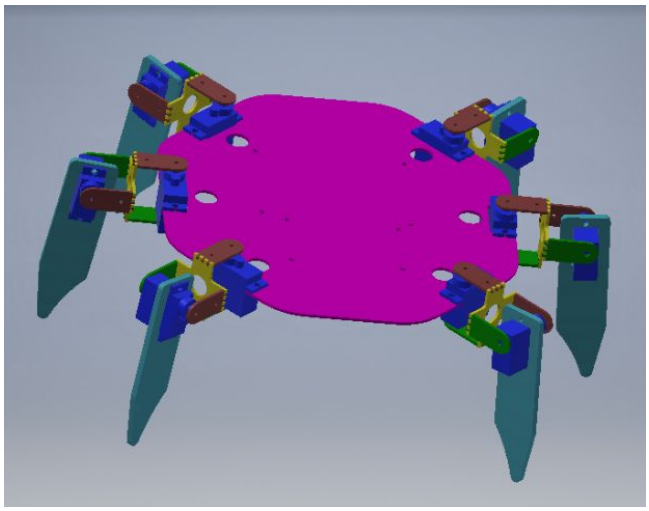


Figure 15: Inventor Assembly Front

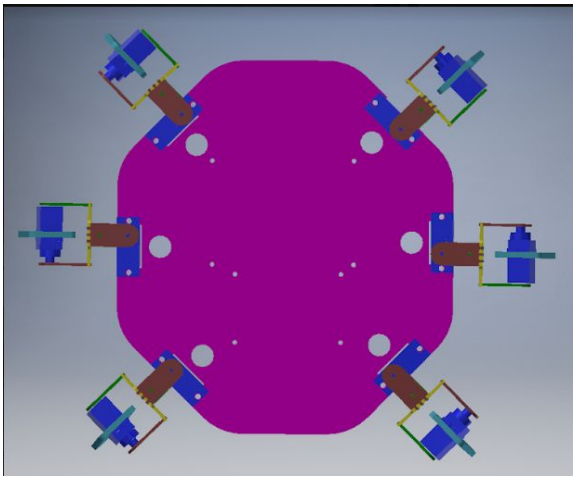


Figure 16: Inventor Assembly Top

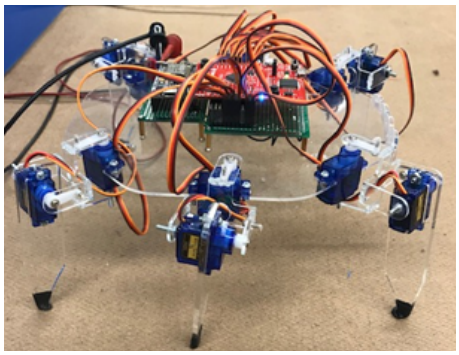


Figure 17: Final Build (Side)

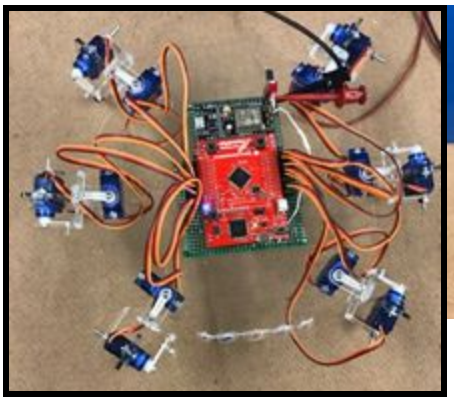


Figure 18: Final Build (Top)

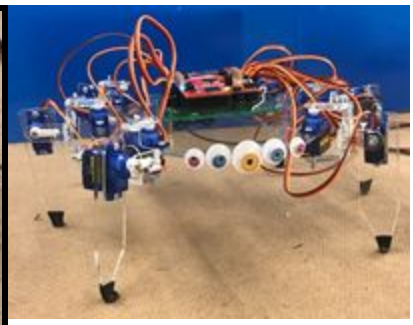


Figure 19: Final Build (Front)

Standards used

ISO/IEC 9899:1999 for C programming language
IEEE 802.11 b/g/n for Wireless Communications
Universal Asynchronous Receiver/Transmitter
Pulse Width Modulation

Cost Analysis/Bill of Materials

The cost to build the device takes into consideration every individual component and necessary material that was used to complete each step of the design from the planning phase up to the build and testing phases. The total bill of materials used to develop the device is included below.

Table 1: Total Bill of Materials Per-Component

Component/ Material	***Quantity***	***Vendor***	***Cost ($)***
Miuzei SG90 Servo Motors	12 (+ Extras)	Amazon.com	34.58
TM4C123GH6PM Microcontroller	1	Mouser	13.45
Node-MCU V1 Wifi Module	1	Amazon.com	8.39
4-40 Machine Bolts and Nuts	30 each (+ Extras)	Raby's Ace	12.44
6 x 8 cm Prototyping PCB	1 (+ Extras)	Amazon.com	5.99
4 x 6 cm Prototyping PCB	1(+ Extras)	Amazon.com	6.99
M2 Standoff Kit	1	Amazon.com	8.99
1/16 in Thick 16x24 in Acrylic Sheet	1	ePlastics.com	8.49
1/8 in Thick 16x24 in Acrylic Sheet	1	ePlastics.com	21.69
Two-Part Quick Setting Epoxy	1	Walmart	5.35
3/8 in Heat Shrink Tubing	1	Jameco	1.75
Colored Googly Eyes	1	Dollar Tree	1.00
Total Cost			*129.11*

The total cost to the students with the resources that SUNY Oswego's ECE Department provides.

Table 2: Total Student Cost Per-Component

Component/ Material	***Quantity***	***Vendor***	***Cost ($)***
Miuzei SG90 Servo Motors	12 (+ Extras)	Amazon.com	34.58
TM4C123GH6PM Microcontroller	1	Provided	--
Node-MCU V1 Wifi Module	1	Amazon.com	8.39
4-40 Machine Bolts and Nuts	30 each (+ Extras)	Raby's Ace	12.44
6 x 8 cm Prototyping PCB	1	Provided	--
4 x 6 cm Prototyping PCB	1	Provided	--
M2 Standoff Kit	1	Amazon.com	8.99
1/16 in Thick 16x24 in Acrylic Sheet	1	Provided	--
1/8 in Thick 16x24 in Acrylic Sheet	1	Provided	--
Two-Part Quick Setting Epoxy	1	Walmart	5.35
3/8 in Heat Shrink Tubing	1	Provided	--
Colored Googly Eyes	1	Dollar Tree	1.00
Total Cost			*70.75*

The operational costs of the walking robot are only that of the cost of electricity that powers the bench power supply.

Hazards & Failure Analysis

As with any solution in development to solve a set of problems, it is the responsibility of the designing engineer to uphold the safety of the public and environment when making any design decision that affects the end-product and how it will be used or perceived by consumers.

Safety was a key feature in the design of the walking robot. There are no sharp edges on the acrylic pieces, and the exposed screws have all be covered with hot glue to prevent any injuries.

The potential for failure is greatest in the joints. Each of the joints is held together with glue, which breaks under high stress.

ABET Learning Outcome

Throughout this project, the take-away from the design process consists of applied knowledge of many of the outcomes outlined by ABET for engineering students required to train as a professional. Various aspects of this design and development process coordinate with these outcomes in many ways.

Engineering Learning Outcomes

1. *an ability to identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics*
    a. The ECE capstone class is design to get students to identify and solve a problem. In this situation, the problem was to create a walking robot. This was solved using principles and skill involved in math, science, and engineering. These skills included soldering, coding and circuitry among countless others. Without the knowledge gained in classes from the ECE major it would have been nearly impossible to complete this project.
2. *an ability to apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors*
    a. Several parts of this project could be considered dangerous if not taken care of. For example, a large amount of voltage could be considered dangerous. This robot runs on 5v, a relatively low amount of voltage which is safe for those working with the project. Several screws are sharp and could conflict pain. These screws have been either filed down or had caps put over them to ensure the safety of those near it. The safety of those working on the project and others who may come in contact it was an important of this project and was considered

3. *an ability to communicate effectively with a range of audiences*
    a. The ability to communicate with a range of audiences was very important in this project. Not only did the team have to communicate with each other, but reports had to be written at every step which explained the project to those who had no part in it. The group made several reports, including this final report, to explain the project in a manner that can be understood by a large range of audiences.
4. *an ability to recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts*
    a. This project can be built upon to eventually create a robot that could help the world. However, as the project is now, the ability to make informed judgements considering global and societal contexts was irrelevant.
5. *an ability to function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives*
    a. One of the very first things that was completed on this project was the creation of a task list. This included each step of the process that was to be completed. The use of a task list allowed the group to create and complete goals. The group that designed this robot consisted of three group members. Each group member worked towards one cohesive finished project but took on smaller tasks to get to that goal. For example, one group member was working on the code itself while another soldered together a circuit for the code to run on. This allowed for the project to be completed in a timely manner. The ability to function well on a team and to follow a set task list allowed for the project to be completed effectively.
6. *an ability to develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions*
    a. Experimentation was used at every step of the process in building this robot. The laser printed legs were one part of the project that took a lot of experimentation.

Initially, the legs were very tall with one point at the bottom. This leg was glued to a servo for initial testing. This leg design did not allow the robot to stand on its own because it was too tall and because the single point caused the leg to slip out from underneath the robot. The leg was then modified to hold one servo near the top, and was shortened in length. This allowed a better connection between the leg and the servo. Eventually, after much experimentation and roughly 8 redesign, the leg which is used in the final project was created. The ability to experiment, gather data, and adjust the project accordingly was very important for this project.

7. *an ability to acquire and apply new knowledge as needed, using appropriate learning strategies*
    a. Through code composer labs, the group was able to learn the necessary commands to program the Tiva-C microcontroller. Other knowledge that may have been needed was requested by faculty or researched separately.

Conclusions and Discussions

The design goal of this walking robot was to walk forward, backward, turn left and right all while keeping its balance. That goal was accomplished, and even expanded upon. For example, the introduction of a WiFi based control system and the two waving functions.

Future Applications:

The design as it currently stands is intended for educational and entertainment uses.

Future Improvements:

1. Time and budget did not permit the inclusion of a battery system to make the system more portable. The use of a battery with a high energy density but a low weight would be ideal, something like a Lithium-Ion or Lithium-Polymer battery pack would work well.
2. Editing the movement methods to make the robot walk more naturally. It currently walks in a very robotic and rigid way. A basic kinematic model could be used to speed up the process of modifying the movement methods.

3. The joints between each servo on the leg are currently held together with glue, which is prone to breaking. A redesign of the joints such that they can be held together with traditional mounting hardware like nuts and bolts would eliminate the problem.

References

[1]B. Baxter, "Tripod-walking Gait," The Tripod Gait as a Model for Robots. [Online]. Available: https://tripodgaitasamodelforrobots.weebly.com/tripod-walking-gait.html.

[2]"Tiva™ TM4C123GH6PM Microcontroller," ti.com. [Online]. Available: http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf.

[3]"SG90 Data Sheet." [Online]. Available: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf.

[4]"NodeMCU ESP8266 ESP-12E WiFi Development Board." [Online]. Available: https://einstronic.com/wp-content/uploads/2017/06/NodeMCU-ESP8266-ESP-12E-Catalogue.pdf.

Appendices

A-1) Servo Code

```
*
* servo.c
* Library based off the TIVA C workshop to control the SG90 micro servos
*
*  Created on: Sep 27, 2018
*      Author: jross5
*/

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/pwm.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/rom.h"

#define PWM_FREQUENCY 55

uint32_t getPWMFrequency(){
      return PWM_FREQUENCY;
}

void delayMS(uint32_t ms)
{
      SysCtlDelay((SysCtlClockGet() / (3 * 1000)) * ms);
}

uint32_t getPWMClock(){
      return (SysCtlClockGet() / 64);
}

uint32_t getLoad(uint32_t ui32PWMClock, uint32_t ui32PWMFreq){
      return ((ui32PWMClock / ui32PWMFreq) - 1);
}
void PWM_Enable(){
      SysCtlPWMClockSet(SYSCTL_PWMDIV_64);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
}
void GPIOPWM_Enable(){
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
```

```
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
}

//-------------------------------------------------------------------------------
------------------------------
//M0 Initilization Methods
//-------------------------------------------------------------------------------
------------------------------
void M0PWM0_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_6);
      GPIOPinConfigure(GPIO_PB6_M0PWM0);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT, true);
      PWMGenEnable(PWM0_BASE, PWM_GEN_0);
}
void M0PWM1_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_7);
      GPIOPinConfigure(GPIO_PB7_M0PWM1);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, true);
      PWMGenEnable(PWM0_BASE, PWM_GEN_0);
}
void M0PWM2_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_4);
      GPIOPinConfigure(GPIO_PB4_M0PWM2);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_1, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_2_BIT, true);
      PWMGenEnable(PWM0_BASE, PWM_GEN_1);
}
void M0PWM3_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_5);
      GPIOPinConfigure(GPIO_PB5_M0PWM3);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_1, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_3_BIT, true);
      PWMGenEnable(PWM0_BASE, PWM_GEN_1);
}
void M0PWM4_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_4);
      GPIOPinConfigure(GPIO_PE4_M0PWM4);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT, true);
      PWMGenEnable(PWM0_BASE, PWM_GEN_2);
}
void M0PWM5_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_5);
      GPIOPinConfigure(GPIO_PE5_M0PWM5);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
```

```
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, true);
      PWMGenEnable(PWM0_BASE, PWM_GEN_2);
}
void M0PWM6_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_4);
      GPIOPinConfigure(GPIO_PC4_M0PWM6);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
      PWMGenEnable(PWM0_BASE, PWM_GEN_3);
}
void M0PWM7_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_5);
      GPIOPinConfigure(GPIO_PC5_M0PWM7);
      PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, ui32Load);
      PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
   PWMGenEnable(PWM0_BASE, PWM_GEN_3);
}

//-------------------------------------------------------------------------------
------------------------------
//M1 Initilization Methods
//-------------------------------------------------------------------------------
------------------------------
void M1PWM0_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
      GPIOPinConfigure(GPIO_PD0_M1PWM0);
      PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_0);
}
void M1PWM1_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_1);
      GPIOPinConfigure(GPIO_PD1_M1PWM1);
      PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_1_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_0);
}
void M1PWM2_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_6);
      GPIOPinConfigure(GPIO_PA6_M1PWM2);
      PWMGenConfigure(PWM1_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_2_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_1);
}
void M1PWM3_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_7);
      GPIOPinConfigure(GPIO_PA7_M1PWM3);
      PWMGenConfigure(PWM1_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN);
```

```
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_3_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_1);
}
void M1PWM4_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);
      GPIOPinConfigure(GPIO_PF0_M1PWM4);
   PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_4_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_2);
}
void M1PWM5_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_1);
      GPIOPinConfigure(GPIO_PF1_M1PWM5);
      PWMGenConfigure(PWM1_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_2, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_5_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_2);
}
void M1PWM6_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
      GPIOPinConfigure(GPIO_PF2_M1PWM6);
      PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_3);
}
void M1PWM7_Init(uint32_t ui32Load){
      GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
      GPIOPinConfigure(GPIO_PF3_M1PWM7);
      PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN);
      PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, ui32Load);
      PWMOutputState(PWM1_BASE, PWM_OUT_7_BIT, true);
      PWMGenEnable(PWM1_BASE, PWM_GEN_3);
}
//-------------------------------------------------------------------------------
------------------------------
//M0 Position Methods
//-------------------------------------------------------------------------------
------------------------------
//M0 PWM0 Pin PB6 position method
void M0PWM0_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
      uint8_t ui8Adjust;

      if (ui8Degrees > 120)
      {
      ui8Adjust = 140;
      }
      else
      {
      ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
```

```
	}
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
}

//M0 PWM1 Pin PB7 position method
void M0PWM1_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, ui8Adjust * ui32Load / 1000);
}

//M0 PWM2 Pin PB4 position method
void M0PWM2_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_2, ui8Adjust * ui32Load / 1000);
}
//M0 PWM3 Pin PB5 position method
void M0PWM3_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_3, ui8Adjust * ui32Load / 1000);
}
//M0 PWM4 Pin PE4 position method
```

```
void M0PWM4_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
      uint8_t ui8Adjust;

      if (ui8Degrees > 120)
      {
      ui8Adjust = 140;
      }
      else
      {
      ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
      }
      PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, ui8Adjust * ui32Load / 1000);
}
//M0 PWM5 Pin PE5 position method
void M0PWM5_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
      uint8_t ui8Adjust;

      if (ui8Degrees > 120)
      {
      ui8Adjust = 140;
      }
      else
      {
      ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
      }
      PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, ui8Adjust * ui32Load / 1000);
}
//M0 PWM6 Pin PC4 position method
void M0PWM6_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
      uint8_t ui8Adjust;

      if (ui8Degrees > 120)
      {
      ui8Adjust = 140;
      }
      else
      {
      ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
      }
      PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, ui8Adjust * ui32Load / 1000);
}
//M0 PWM7 Pin PC5 position method
void M0PWM7_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
      uint8_t ui8Adjust;

      if (ui8Degrees > 120)
```

```
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, ui8Adjust * ui32Load / 1000);
}

//-------------------------------------------------------------------------------
------------------------------
//M1 Position Methods
//-------------------------------------------------------------------------------
------------------------------
//M1 PWM0 Pin PD0 position method
void M1PWM0_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust * ui32Load / 1000);
}

//M1 PWM1 Pin PD1 position method
void M1PWM1_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, ui8Adjust * ui32Load / 1000);
}

//M1 PWM2 Pin PA6 position method
void M1PWM2_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;
```

```
	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, ui8Adjust * ui32Load / 1000);
}
//M1 PWM3 Pin PA7 position method
void M1PWM3_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_3, ui8Adjust * ui32Load / 1000);
}
//M1 PWM4 Pin PF0 position method
void M1PWM4_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_4, ui8Adjust * ui32Load / 1000);
}
//M1 PWM5 Pin PF1 position method
void M1PWM5_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
```

```
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_5, ui8Adjust * ui32Load / 1000);
}
//M1 PWM6 Pin PF2 position method
void M1PWM6_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, ui8Adjust * ui32Load / 1000);
}
//M1 PWM7 Pin PF3 position method
void M1PWM7_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load) // Servo
position from 0 to 120 degrees
{
	uint8_t ui8Adjust;

	if (ui8Degrees > 120)
	{
	ui8Adjust = 140;
	}
	else
	{
	ui8Adjust = 56.0 + ((float) ui8Degrees * (140.0 - 56.0)) / 120.0;
	}
	PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, ui8Adjust * ui32Load / 1000);
}
```

A-2) Servo Header File

```
/*
 * servo.h
 *
 *  Created on: Sep 27, 2018
 *      Author: jross5
 */

#ifndef SERVO_H_
#define SERVO_H_

extern void delayMS(uint32_t ms);

extern uint32_t getPWMFrequency();
extern uint32_t getPWMClock();
extern uint32_t getLoad(uint32_t ui32PWMClock, uint32_t ui32PWMFreq);

extern void PWM_Enable();
extern void GPIOPWM_Enable();

extern void M0PWM0_Init(uint32_t ui32Load);
extern void M0PWM1_Init(uint32_t ui32Load);
extern void M0PWM2_Init(uint32_t ui32Load);
extern void M0PWM3_Init(uint32_t ui32Load);
extern void M0PWM4_Init(uint32_t ui32Load);
extern void M0PWM5_Init(uint32_t ui32Load);
extern void M0PWM6_Init(uint32_t ui32Load);
extern void M0PWM7_Init(uint32_t ui32Load);

extern void M1PWM0_Init(uint32_t ui32Load);
extern void M1PWM1_Init(uint32_t ui32Load);
extern void M1PWM2_Init(uint32_t ui32Load);
extern void M1PWM3_Init(uint32_t ui32Load);
extern void M1PWM4_Init(uint32_t ui32Load);
extern void M1PWM5_Init(uint32_t ui32Load);
extern void M1PWM6_Init(uint32_t ui32Load);
extern void M1PWM7_Init(uint32_t ui32Load);

extern void M0PWM0_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M0PWM1_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M0PWM2_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M0PWM3_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M0PWM4_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M0PWM5_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M0PWM6_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M0PWM7_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);

extern void M1PWM0_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M1PWM1_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M1PWM2_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M1PWM3_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
```

```
extern void M1PWM4_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M1PWM5_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M1PWM6_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);
extern void M1PWM7_PositionServo(uint8_t ui8Degrees, uint32_t ui32Load);


#endif /* SERVO_H_ */
```

A-3) Legs Code

```
/*
 * legs.c
 * Library based off the servo.c that abstracts the control of servos
 *  Created on: Oct 9, 2018
 *      Author: jross5
 */

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
#include "driverlib/pwm.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/rom.h"
#include "servo.h"

//-------------------------------------------------------------------------------
-------
//INITIALIZATION METHODS
//-------------------------------------------------------------------------------
-------

//Initialize each PWM pin
void All_PWM_Init(){
    uint32_t ui32PWM_Freq = getPWMFrequency();
    GPIOPWM_Enable();
    PWM_Enable();

    // Unlock the Pin PF0 and Set the Commit Bit
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR)   |= 0x01;

    uint32_t ui32PWMClock = getPWMClock();
    uint32_t ui32Load = getLoad(ui32PWMClock, ui32PWM_Freq);

    M0PWM0_Init(ui32Load); //PB6
    M0PWM1_Init(ui32Load); //PB7
    M0PWM2_Init(ui32Load); //PB4
    M0PWM3_Init(ui32Load); //PB5
    M0PWM4_Init(ui32Load); //PE4
    M0PWM5_Init(ui32Load); //PE5
    M0PWM6_Init(ui32Load); //PC4
    M0PWM7_Init(ui32Load); //PC5

    M1PWM0_Init(ui32Load); //PD0
    M1PWM1_Init(ui32Load); //PD1
```

```
    M1PWM2_Init(ui32Load); //PA6
    M1PWM3_Init(ui32Load); //PA7
    M1PWM4_Init(ui32Load); //PF0
    M1PWM5_Init(ui32Load); //PF1
    M1PWM6_Init(ui32Load); //PF2
    M1PWM7_Init(ui32Load); //PF3
}

//Initialize each pin used for the hex's Legs
void Legs_Init(uint32_t ui32Load){
    GPIOPWM_Enable();
    PWM_Enable();
    // Unlock the Pin PF0 and Set the Commit Bit
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR)   |= 0x01;

    M0PWM0_Init(ui32Load); //PB6
    M0PWM1_Init(ui32Load); //PB7
    M0PWM2_Init(ui32Load); //PB4
    M0PWM3_Init(ui32Load); //PB5
    M0PWM4_Init(ui32Load); //PE4
    M0PWM5_Init(ui32Load); //PE5
    M0PWM7_Init(ui32Load); //PC5

    M1PWM2_Init(ui32Load); //PA6
    M1PWM3_Init(ui32Load); //PA7
    M1PWM4_Init(ui32Load); //PF0
    M1PWM6_Init(ui32Load); //PF2
    M1PWM7_Init(ui32Load); //PF3

}

//-------------------------------------------------------------------------
-------
//LEFT LEG POSITION METHODS
//-------------------------------------------------------------------------
-------

//Front Left Leg position method
//L1F - pin PB5
//L2F - pin PE4
void LegLF_Position(uint8_t ui8L1Degrees,uint8_t ui8L2Degrees,uint32_t
ui32Load){
    M0PWM3_PositionServo(ui8L1Degrees, ui32Load);
    M0PWM4_PositionServo(ui8L2Degrees, ui32Load);
}
//Middle Left Leg position method
//L1M - pin PE5
//L2M - pin PB4
void LegLM_Position(uint8_t ui8L1Degrees,uint8_t ui8L2Degrees,uint32_t
ui32Load){
    M0PWM5_PositionServo(ui8L1Degrees, ui32Load);
    M0PWM2_PositionServo(ui8L2Degrees, ui32Load);
}
```

```
//Back Left Leg position method
//L1B - pin PA6
//L2B - pin PA7
void LegLB_Position(uint8_t ui8L1Degrees,uint8_t ui8L2Degrees,uint32_t
ui32Load){
    M1PWM2_PositionServo(ui8L1Degrees, ui32Load);
    M1PWM3_PositionServo(ui8L2Degrees, ui32Load);
}

//-------------------------------------------------------------------------
-------
//RIGHT LEG POSITION METHODS
//-------------------------------------------------------------------------
-------

//Front Right Leg position method
//R1F - pin PF2
//R2F - pin PF3
void LegRF_Position(uint8_t ui8R1Degrees,uint8_t ui8R2Degrees,uint32_t
ui32Load){
    M1PWM6_PositionServo(ui8R1Degrees, ui32Load);
    M1PWM7_PositionServo(ui8R2Degrees, ui32Load);
}
//Middle Right Leg position method
//R1M - pin PF0
//R2M - pin PC5
void LegRM_Position(uint8_t ui8R1Degrees,uint8_t ui8R2Degrees,uint32_t
ui32Load){
    M1PWM4_PositionServo(ui8R1Degrees, ui32Load);
    M0PWM7_PositionServo(ui8R2Degrees, ui32Load);
}
//Back right Leg position method
//R1B - pin PB7
//R2B - pin PB6
void LegRB_Position(uint8_t ui8R1Degrees,uint8_t ui8R2Degrees,uint32_t
ui32Load){
    M0PWM1_PositionServo(ui8R1Degrees, ui32Load);
    M0PWM0_PositionServo(ui8R2Degrees, ui32Load);
}

//-------------------------------------------------------------------------
-------
// Custom Position Methods
//-------------------------------------------------------------------------
-------
//Set every servos on body to 60 and servos on legs to 0 degrees. Standing
neutrally
void Neutral_Stance(uint32_t ui32Load){
    LegLF_Position(60,0,ui32Load);
    LegLM_Position(60,0,ui32Load);
    LegLB_Position(60,0,ui32Load);

    LegRF_Position(60,0,ui32Load);
    LegRM_Position(60,0,ui32Load);
```

```
    LegRB_Position(60,0,ui32Load);
}

//Walk Forward one cycle
void forward(uint32_t ui32Load, uint32_t delay){
    LegLF_Position(20,45,ui32Load);
    LegLB_Position(20,45,ui32Load);
    LegRM_Position(110,45,ui32Load);
    delayMS(delay);
    LegLF_Position(20,0,ui32Load);
    LegLB_Position(20,0,ui32Load);
    LegRM_Position(110,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,20,ui32Load);
    LegRB_Position(60,20,ui32Load);
    LegLM_Position(60,20,ui32Load);
    delayMS(delay);
    LegLF_Position(60,0,ui32Load);
    LegLB_Position(60,0,ui32Load);
    LegRM_Position(60,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,0,ui32Load);
    LegRB_Position(60,0,ui32Load);
    LegLM_Position(60,0,ui32Load);

    LegRF_Position(120,45,ui32Load);
    LegRB_Position(120,45,ui32Load);
    LegLM_Position(0,45,ui32Load);
    delayMS(delay);
    LegRF_Position(120,0,ui32Load);
    LegRB_Position(120,0,ui32Load);
    LegLM_Position(0,0,ui32Load);
    delayMS(delay);
    LegLF_Position(60,20,ui32Load);
    LegLB_Position(60,20,ui32Load);
    LegRM_Position(60,20,ui32Load);
    delayMS(delay);
    LegRF_Position(60,0,ui32Load);
    LegRB_Position(60,0,ui32Load);
    LegLM_Position(60,0,ui32Load);
    delayMS(delay);
    LegLF_Position(60,0,ui32Load);
    LegLB_Position(60,0,ui32Load);
    LegRM_Position(60,0,ui32Load);
}

//Walk backward one cycle
void backward(uint32_t ui32Load, uint32_t delay){
    LegLF_Position(120,45,ui32Load);
    LegLB_Position(120,45,ui32Load);
    LegRM_Position(0,45,ui32Load);
    delayMS(delay);
    LegLF_Position(120,0,ui32Load);
    LegLB_Position(120,0,ui32Load);
```

```
    LegRM_Position(0,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,20,ui32Load);
    LegRB_Position(60,20,ui32Load);
    LegLM_Position(60,20,ui32Load);
    delayMS(delay);
    LegLF_Position(60,0,ui32Load);
    LegLB_Position(60,0,ui32Load);
    LegRM_Position(60,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,0,ui32Load);
    LegRB_Position(60,0,ui32Load);
    LegLM_Position(60,0,ui32Load);

    LegRF_Position(0,45,ui32Load);
    LegRB_Position(0,45,ui32Load);
    LegLM_Position(120,45,ui32Load);
    delayMS(delay);
    LegRF_Position(0,0,ui32Load);
    LegRB_Position(0,0,ui32Load);
    LegLM_Position(120,0,ui32Load);
    delayMS(delay);
    LegLF_Position(60,20,ui32Load);
    LegLB_Position(60,20,ui32Load);
    LegRM_Position(60,20,ui32Load);
    delayMS(delay);
    LegRF_Position(60,0,ui32Load);
    LegRB_Position(60,0,ui32Load);
    LegLM_Position(60,0,ui32Load);
    delayMS(delay);
    LegLF_Position(60,0,ui32Load);
    LegLB_Position(60,0,ui32Load);
    LegRM_Position(60,0,ui32Load);
}
//Turn left one cycle
void turn_Left(uint32_t ui32Load, uint32_t delay){
    LegLF_Position(120,20,ui32Load);
    LegLB_Position(120,20,ui32Load);
    LegRM_Position(120,20,ui32Load);
    delayMS(delay);
    LegLF_Position(120,0,ui32Load);
    LegLB_Position(120,0,ui32Load);
    LegRM_Position(120,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,20,ui32Load);
    LegRB_Position(60,20,ui32Load);
    LegLM_Position(60,20,ui32Load);
    delayMS(delay);
    LegLF_Position(60,0,ui32Load);
    LegLB_Position(60,0,ui32Load);
    LegRM_Position(60,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,0,ui32Load);
    LegRB_Position(60,0,ui32Load);
```

```
    LegLM_Position(60,0,ui32Load);
    delayMS(delay);
}

//Turn Right one cycle
void turn_Right(uint32_t ui32Load, uint32_t delay){
    LegLF_Position(0,20,ui32Load);
    LegLB_Position(0,20,ui32Load);
    LegRM_Position(0,20,ui32Load);
    delayMS(delay);
    LegLF_Position(0,0,ui32Load);
    LegLB_Position(0,0,ui32Load);
    LegRM_Position(0,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,20,ui32Load);
    LegRB_Position(60,20,ui32Load);
    LegLM_Position(60,20,ui32Load);
    delayMS(delay);
    LegLF_Position(60,0,ui32Load);
    LegLB_Position(60,0,ui32Load);
    LegRM_Position(60,0,ui32Load);
    delayMS(delay);
    LegRF_Position(60,0,ui32Load);
    LegRB_Position(60,0,ui32Load);
    LegLM_Position(60,0,ui32Load);
    delayMS(delay);
}

void wave_one(uint32_t ui32Load){
    LegRF_Position(0,120,ui32Load);
    delayMS(250);
    LegRF_Position(120,120,ui32Load);
    delayMS(250);
    LegRF_Position(0,120,ui32Load);
}

void wave_two(uint32_t ui32Load){
    LegRM_Position(100,0,ui32Load);
    LegLM_Position(20,0,ui32Load);

    LegRB_Position(20,0,ui32Load);
    LegLB_Position(100,0,ui32Load);

    LegRF_Position(0,120,ui32Load);
    LegLF_Position(120,120,ui32Load);
    delayMS(250);
    LegRF_Position(120,120,ui32Load);
    LegLF_Position(0,120,ui32Load);
    delayMS(250);
    LegRF_Position(0,120,ui32Load);
    LegLF_Position(120,120,ui32Load);
}
```

A-4) Legs Header File

```
/*
 * legs.h
 *
 *  Created on: Oct 9, 2018
 *      Author: jross5
 */

#ifndef LEGS_H_
#define LEGS_H_

extern void All_PWM_Init();
extern void Legs_Init(uint32_t ui32Load);

extern void LegLF_Position(uint8_t ui8L1Degrees,uint8_t ui8L2Degrees,uint32_t
ui32Load);
extern void LegLM_Position(uint8_t ui8L1Degrees,uint8_t ui8L2Degrees,uint32_t
ui32Load);
extern void LegLB_Position(uint8_t ui8L1Degrees,uint8_t ui8L2Degrees,uint32_t
ui32Load);

extern void LegRF_Position(uint8_t ui8R1Degrees,uint8_t ui8R2Degrees,uint32_t
ui32Load);
extern void LegRM_Position(uint8_t ui8R1Degrees,uint8_t ui8R2Degrees,uint32_t
ui32Load);
extern void LegRB_Position(uint8_t ui8R1Degrees,uint8_t ui8R2Degrees,uint32_t
ui32Load);

extern void Neutral_Stance(uint32_t ui32Load);
extern void forward(uint32_t ui32Load, uint32_t delay);
extern void backward(uint32_t ui32Load, uint32_t delay);
extern void turn_Right(uint32_t ui32Load, uint32_t delay);
extern void turn_Left(uint32_t ui32Load, uint32_t delay);
extern void wave_one(uint32_t ui32Load);
extern void wave_two(uint32_t ui32Load);

#endif /* LEGS_H_ */
```

A-5) UART Code

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"

//Enable UART1 and UART Interrupt
//Reciever PB0
//Transmitter PB1
void UART1_Init(){
      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
      GPIOPinConfigure(GPIO_PB0_U1RX);
    GPIOPinConfigure(GPIO_PB1_U1TX);
    GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
      IntMasterEnable(); //enable processor interrupts
    IntEnable(INT_UART1); //enable the UART interrupt
    UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT); //only enable RX and
TX interrupts
}
//Enable UART0 for serial communication via USB
//Reciever PA0
//Transmitter PA1
void UART0_Init(){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
}
```

A-6) UART Header File

```
/*
 * uartcomm.h
 *
 *  Created on: Nov 4, 2018
 *      Author: Justin
 */

#ifndef UARTCOMM_H_
#define UARTCOMM_H_


extern void UART1_Init();
extern void UART0_Init();

#endif /* UARTCOMM_H_ */
```

A-7) Main Code

```
//Main Code for Tiva
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "servo.h"
#include "legs.h"
#include "uartcomm.h"


char last_Command = 'S';
uint8_t cont_Command = 0;
uint8_t fast_Command = 0;

void UARTIntHandler(void){
    uint32_t ui32Status;
    char tempChar;

    ui32Status = UARTIntStatus(UART1_BASE, true); //get interrupt status

    UARTIntClear(UART1_BASE, ui32Status); //clear the asserted interrupts

    tempChar = UARTCharGet(UART1_BASE);
    if(tempChar == 'C'){
        if(cont_Command == 1){
            cont_Command = 0;
        }else{
            cont_Command = 1;
        }
    }else if(tempChar == 'F'){
        if(fast_Command == 1){
            fast_Command = 0;
        }else{
            fast_Command = 1;
        }
    }else {
        last_Command = tempChar;
    }
}

 int main(void) {
    uint32_t ui32PWM_Freq = getPWMFrequency();
```

```
    SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_16MHZ);

    uint32_t ui32PWMClock = getPWMClock();
    uint32_t ui32Load = getLoad(ui32PWMClock, ui32PWM_Freq);
    Legs_Init(ui32Load);
    UART1_Init();

    while (1){
        switch (last_Command){
            case 'S':
                Neutral_Stance(ui32Load);
                break;
            case 'U':
                if(fast_Command == 0){
                    forward(ui32Load, 500);
                }else {
                    forward(ui32Load, 100);
                }

                if(cont_Command == 0){last_Command = 'S';}
                break;
            case 'D':
                if(fast_Command == 0){
                    backward(ui32Load, 500);
                }else {
                    backward(ui32Load, 100);
                }
             if(cont_Command==0){last_Command = 'S';}
                break;
            case 'L':
                if(fast_Command == 0){
                    turn_Left(ui32Load, 500);
                }else {
                    turn_Left(ui32Load, 100);
                }
               if(cont_Command==0){last_Command = 'S';}
                break;
            case 'R':
                if(fast_Command == 0){
                    turn_Right(ui32Load, 500);
                }else {
                    turn_Right(ui32Load, 100);
                }
              if(cont_Command==0){last_Command = 'S';}
                break;
            case 'H':
                wave_one(ui32Load);
                wave_one(ui32Load);
                last_Command = 'S';
                break;
            case 'W':
                wave_two(ui32Load);
                wave_two(ui32Load);
```

```
                last_Command = 'S';
                break;
            default:
                Neutral_Stance(ui32Load);
                break;
        }
    }

}
```

A-8) tm4c123gh6pm_startup_ccs.c

```
//*****************************************************************************
//
// Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2011-2014 Texas Instruments Incorporated.  All rights
reserved.
// Software License Agreement
//
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
//*****************************************************************************

#include <stdint.h>

//*****************************************************************************
//
// Forward declaration of the default fault handlers.
//
//*****************************************************************************
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

//*****************************************************************************
//
// External declaration for the reset handler that is to be called when the
// processor is started
//
//*****************************************************************************
extern void _c_int00(void);

extern void UARTIntHandler(void);
//*****************************************************************************
//
// Linker variable that marks the top of the stack.
```

```
//
//*****************************************************************************
extern uint32_t __STACK_TOP;

//*****************************************************************************
//
// External declarations for the interrupt handlers used by the application.
//
//*****************************************************************************
// To be added by user

//*****************************************************************************
//
// The vector table.  Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000 or at the start of
// the program if located at a start address other than 0.
//
//*****************************************************************************
#pragma DATA_SECTION(g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_TOP),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // SVCall handler
    IntDefaultHandler,                      // Debug monitor handler
    0,                                      // Reserved
    IntDefaultHandler,                      // The PendSV handler
    IntDefaultHandler,                      // The SysTick handler
    IntDefaultHandler,                      // GPIO Port A
    IntDefaultHandler,                      // GPIO Port B
    IntDefaultHandler,                      // GPIO Port C
    IntDefaultHandler,                      // GPIO Port D
    IntDefaultHandler,                      // GPIO Port E
    IntDefaultHandler,                      // UART0 Rx and Tx
    UARTIntHandler,                         // UART1 Rx and Tx
    IntDefaultHandler,                      // SSI0 Rx and Tx
    IntDefaultHandler,                      // I2C0 Master and Slave
    IntDefaultHandler,                      // PWM Fault
    IntDefaultHandler,                      // PWM Generator 0
    IntDefaultHandler,                      // PWM Generator 1
    IntDefaultHandler,                      // PWM Generator 2
    IntDefaultHandler,                      // Quadrature Encoder 0
    IntDefaultHandler,                      // ADC Sequence 0
    IntDefaultHandler,                      // ADC Sequence 1
```

```
IntDefaultHandler,                      // ADC Sequence 2
IntDefaultHandler,                      // ADC Sequence 3
IntDefaultHandler,                      // Watchdog timer
IntDefaultHandler,                      // Timer 0 subtimer A
IntDefaultHandler,                      // Timer 0 subtimer B
IntDefaultHandler,                      // Timer 1 subtimer A
IntDefaultHandler,                      // Timer 1 subtimer B
IntDefaultHandler,                      // Timer 2 subtimer A
IntDefaultHandler,                      // Timer 2 subtimer B
IntDefaultHandler,                      // Analog Comparator 0
IntDefaultHandler,                      // Analog Comparator 1
IntDefaultHandler,                      // Analog Comparator 2
IntDefaultHandler,                      // System Control (PLL, OSC, BO)
IntDefaultHandler,                      // FLASH Control
IntDefaultHandler,                      // GPIO Port F
IntDefaultHandler,                      // GPIO Port G
IntDefaultHandler,                      // GPIO Port H
IntDefaultHandler,                      // UART2 Rx and Tx
IntDefaultHandler,                      // SSI1 Rx and Tx
IntDefaultHandler,                      // Timer 3 subtimer A
IntDefaultHandler,                      // Timer 3 subtimer B
IntDefaultHandler,                      // I2C1 Master and Slave
IntDefaultHandler,                      // Quadrature Encoder 1
IntDefaultHandler,                      // CAN0
IntDefaultHandler,                      // CAN1
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // Hibernate
IntDefaultHandler,                      // USB0
IntDefaultHandler,                      // PWM Generator 3
IntDefaultHandler,                      // uDMA Software Transfer
IntDefaultHandler,                      // uDMA Error
IntDefaultHandler,                      // ADC1 Sequence 0
IntDefaultHandler,                      // ADC1 Sequence 1
IntDefaultHandler,                      // ADC1 Sequence 2
IntDefaultHandler,                      // ADC1 Sequence 3
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // GPIO Port J
IntDefaultHandler,                      // GPIO Port K
IntDefaultHandler,                      // GPIO Port L
IntDefaultHandler,                      // SSI2 Rx and Tx
IntDefaultHandler,                      // SSI3 Rx and Tx
IntDefaultHandler,                      // UART3 Rx and Tx
IntDefaultHandler,                      // UART4 Rx and Tx
IntDefaultHandler,                      // UART5 Rx and Tx
IntDefaultHandler,                      // UART6 Rx and Tx
IntDefaultHandler,                      // UART7 Rx and Tx
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // I2C2 Master and Slave
IntDefaultHandler,                      // I2C3 Master and Slave
```

```
IntDefaultHandler,                      // Timer 4 subtimer A
IntDefaultHandler,                      // Timer 4 subtimer B
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // Timer 5 subtimer A
IntDefaultHandler,                      // Timer 5 subtimer B
IntDefaultHandler,                      // Wide Timer 0 subtimer A
IntDefaultHandler,                      // Wide Timer 0 subtimer B
IntDefaultHandler,                      // Wide Timer 1 subtimer A
IntDefaultHandler,                      // Wide Timer 1 subtimer B
IntDefaultHandler,                      // Wide Timer 2 subtimer A
IntDefaultHandler,                      // Wide Timer 2 subtimer B
IntDefaultHandler,                      // Wide Timer 3 subtimer A
IntDefaultHandler,                      // Wide Timer 3 subtimer B
IntDefaultHandler,                      // Wide Timer 4 subtimer A
IntDefaultHandler,                      // Wide Timer 4 subtimer B
IntDefaultHandler,                      // Wide Timer 5 subtimer A
IntDefaultHandler,                      // Wide Timer 5 subtimer B
IntDefaultHandler,                      // FPU
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // I2C4 Master and Slave
IntDefaultHandler,                      // I2C5 Master and Slave
IntDefaultHandler,                      // GPIO Port M
IntDefaultHandler,                      // GPIO Port N
IntDefaultHandler,                      // Quadrature Encoder 2
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // GPIO Port P (Summary or P0)
IntDefaultHandler,                      // GPIO Port P1
IntDefaultHandler,                      // GPIO Port P2
IntDefaultHandler,                      // GPIO Port P3
IntDefaultHandler,                      // GPIO Port P4
IntDefaultHandler,                      // GPIO Port P5
IntDefaultHandler,                      // GPIO Port P6
IntDefaultHandler,                      // GPIO Port P7
```

```
    IntDefaultHandler,                      // GPIO Port Q (Summary or Q0)
    IntDefaultHandler,                      // GPIO Port Q1
    IntDefaultHandler,                      // GPIO Port Q2
    IntDefaultHandler,                      // GPIO Port Q3
    IntDefaultHandler,                      // GPIO Port Q4
    IntDefaultHandler,                      // GPIO Port Q5
    IntDefaultHandler,                      // GPIO Port Q6
    IntDefaultHandler,                      // GPIO Port Q7
    IntDefaultHandler,                      // GPIO Port R
    IntDefaultHandler,                      // GPIO Port S
    IntDefaultHandler,                      // PWM 1 Generator 0
    IntDefaultHandler,                      // PWM 1 Generator 1
    IntDefaultHandler,                      // PWM 1 Generator 2
    IntDefaultHandler,                      // PWM 1 Generator 3
    IntDefaultHandler                       // PWM 1 Fault
};

//*****************************************************************************
//
// This is the code that gets called when the processor first starts execution
// following a reset event.  Only the absolutely necessary set is performed,
// after which the application supplied entry() routine is called.  Any fancy
// actions (such as making decisions based on the reset cause register, and
// resetting the bits in that register) are left solely in the hands of the
// application.
//
//*****************************************************************************
void
ResetISR(void)
{
    //
    // Jump to the CCS C initialization routine.  This will enable the
    // floating-point unit as well, so that does not need to be done here.
    //
    __asm("    .global _c_int00\n"
          "    b.w     _c_int00");
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a NMI.  This
// simply enters an infinite loop, preserving the system state for examination
// by a debugger.
//
//*****************************************************************************
static void
NmiSR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
```

```
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a fault
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
FaultISR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives an unexpected
// interrupt.  This simply enters an infinite loop, preserving the system state
// for examination by a debugger.
//
//*****************************************************************************
static void
IntDefaultHandler(void)
{
    //
    // Go into an infinite loop.
    //
    while(1)
    {
    }
}
```

A-9) NodeMCU Code

```
//Code for Hexapod Capstone Controller
//Written for NODEMCU
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include "index.h"

const char *ssid = "CAPSTONE";
const char *password = "SUNY2018";
const int max_connection = 1;

String header = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
String page = MAIN_page;
String request = "";

WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  WiFi.softAP(ssid, password,max_connection);
  IPAddress myIP = WiFi.softAPIP();
  server.begin();
}

void loop() {
    WiFiClient client = server.available();
    if (!client)  {
      return;
    }
    request = client.readStringUntil('\r');

  if ( request.indexOf("STOP") > 0 ) {
                Serial.print('S');
                client.print( header );
                client.print( "Stop was Pressed" );
  }else if ( request.indexOf("UP") > 0 ) {
                Serial.print('U');
                client.print( header );
                client.print( "Up was Pressed" );
   }else if  ( request.indexOf("DOWN") > 0 ) {
                Serial.print('D');
                client.print( header );
                client.print( "Down was Pressed" );
   }else if  ( request.indexOf("LEFT") > 0 ){
                Serial.print('L');
                client.print( header );
                client.print( "Left was Pressed" );
   }else if  ( request.indexOf("RIGHT") > 0 ) {
                Serial.print("R");
                client.print( header );
                client.print( "Right was Pressed" );
```

```
   }else if  ( request.indexOf("HELLO") > 0 ) {
                Serial.print("H");
                client.print( header );
                client.print( "Hello World!" );
   }else if (request.indexOf("WAVE") > 0) {
                Serial.print("W");
                client.print( header );
                client.print( "Hello World!" );
   }else if (request.indexOf("CONT") > 0){
                Serial.print("C");
                client.print( header );
    }else if (request.indexOf("FAST") > 0){
                Serial.print("F");
                client.print( header );
               client.print( "RB" );
   }else {
                client.flush();
                client.print( header );
                client.print( page );
                delay(5);
    }
}
```

A-10)Index.h

```
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
 <head>
 <meta name='viewport' content='width=device-width, initial-scale=1.0'/>
 <meta charset='utf-8'>

<style>

.base-grid{
  display: grid;
  grid-template-columns: auto auto;
  grid-template-rows: auto auto;
}

.grid-container {
  display: grid;
  grid-gap: 10px 10px;
  grid-template-columns: auto auto auto;
  background-color: #FFFFFF;
  padding: 10px;
}

.grid-item {
  background-color: rgba(255, 255, 255, 255);
  font-size: 30px;
  text-align: center;
}
.grid-item2 {
  background-color: rgba(255, 255, 255, 255);
  text-align: left;
 }

.button {
    background-color: #efefef; /* Gray */
    border: 2px solid black;
      border-radius: 8px;
    color: black;
    padding: 10px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 30px;
    margin: 4px 2px;
    cursor: pointer;
}
.panicbutton {
    background-color: #ff0000; /* RED */
    border: 2px solid black;
      border-radius: 8px;
```

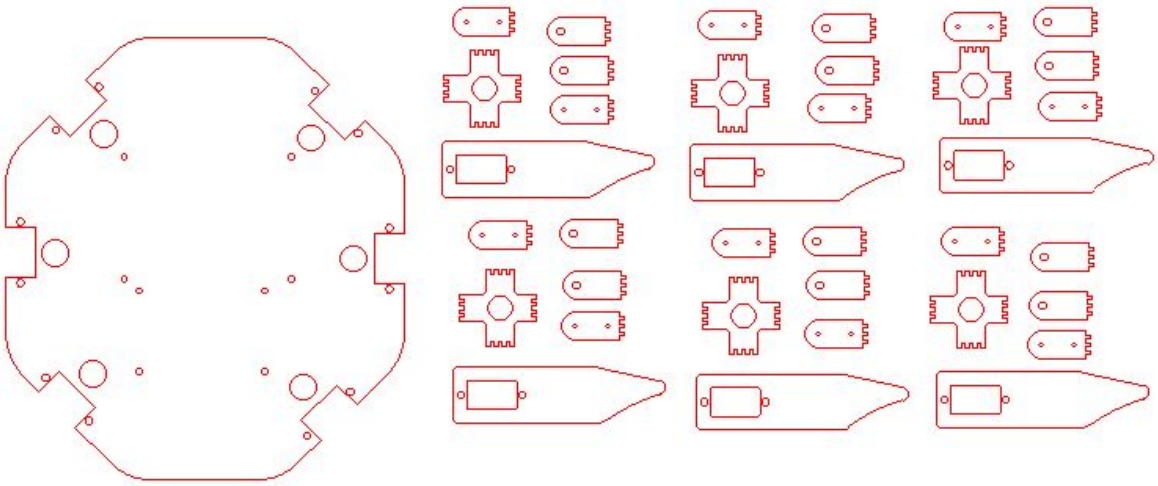
```
    color: white;
    padding: 10px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 30px;
    margin: 4px 2px;
    cursor: pointer;
}
.hellobutton {
    background-color: #00ff00; /* GREEN */
    border: 2px solid black;
      border-radius: 8px;
    color: black;
    padding: 10px;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    font-size: 30px;
    margin: 4px 2px;
    cursor: pointer;
}
.switch {
  position: relative;
  display: inline-block;
  width: 60px;
  height: 34px;
}

/* Hide default HTML checkbox */
.switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

/* The slider */
.slider {
  position: absolute;
  cursor: pointer;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: #ccc;
  border: 2px solid black;
  -webkit-transition: .4s;
  transition: .4s;
}

.slider:before {
  position: absolute;
  content: "";
  height: 26px;
```

```
  width: 26px;
  left: 4px;
  bottom: 4px;
  background-color: white;
  -webkit-transition: .4s;
  transition: .4s;
}

input:checked + .slider {
  background-color: #00ff00;
}

input:focus + .slider {
  box-shadow: 0 0 1px #2196F3;
}

input:checked + .slider:before {
  -webkit-transform: translateX(26px);
  -ms-transform: translateX(26px);
  transform: translateX(26px);
}

/* Rounded sliders */
.slider.round {
  border-radius: 34px;
}

.slider.round:before {
  border-radius: 50%;
}

</style>

<script>
function upClick(){
      ajaxLoad('UP');
}

function leftClick(){
      ajaxLoad('LEFT');
}

function rightClick(){
      ajaxLoad('RIGHT');
}

function downClick(){
      ajaxLoad('DOWN');
}

function stopClick(){
      ajaxLoad('STOP');
}
```

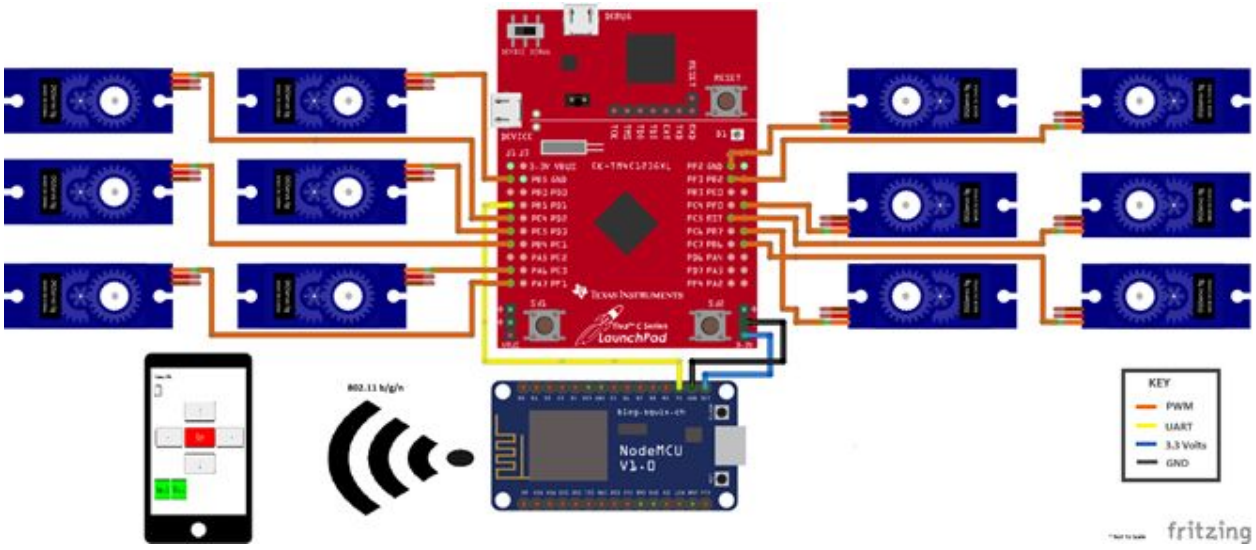
```
function helloClick(){
      ajaxLoad('HELLO');
}
function hello2Click(){
      ajaxLoad('WAVE');
}
function continueClick(){
      ajaxLoad('CONT');
}
function fastClick(){
      ajaxLoad('FAST');
}

var ajaxRequest = null;
if (window.XMLHttpRequest)  { ajaxRequest =new XMLHttpRequest(); }
else                        { ajaxRequest =new
ActiveXObject("Microsoft.XMLHTTP"); }

function ajaxLoad(ajaxURL)
{
  if(!ajaxRequest){ alert("AJAX is not supported."); return; }

  ajaxRequest.open("GET",ajaxURL,true);
  ajaxRequest.onreadystatechange = function()
  {
    if(ajaxRequest.readyState == 4 && ajaxRequest.status==200)
    {
      var ajaxResult = ajaxRequest.responseText;
      document.getElementById("reply").innerHTML = ajaxResult;
    }
  }
  ajaxRequest.send();
}
</script>

<title>Fall 2018 ECE Capstone</title>
</head>

<body>
<div class="grid">
<div class="grid-item2"><b>Continuous Mode</b></div>
<div> <label class="switch">
            <input type="checkbox">
            <span class="slider round" onclick ="continueClick()"></span>
      </label>
</div>
<div class="grid-item2"><b>Fast Mode</b></div>
<div> <label class="switch">
            <input type="checkbox">
            <span class="slider round" onclick ="fastClick()"></span>
      </label>
</div>

</div>
```

```
<div class="grid-container">
  <div class="grid-item"></div>
  <div class="button" onmousedown ="upClick()">&uarr;</div>
  <div class="grid-item"></div>
  <div class="button" onmousedown ="leftClick()">&larr;</div>
  <div class="panicbutton" onmousedown="stopClick()">Stop</div>
  <div class="button" onmousedown ="rightClick()">&rarr;</div>
  <div class="grid-item"></div>
  <div class="button" onmousedown ="downClick()">&darr;</div>
  <div class="grid-item"></div>
</div>

<div class="hellobutton" onmousedown = "helloClick()"> Wave 1</div>
<div class="hellobutton" onmousedown = "hello2Click()"> Wave 2</div>
<p> Last Command Sent </p>
<p id="reply"> Reply Here</p>
</body>
</html>

)=====";
```

A-11)Laser Cutting Diagram



A-12) Wiring Diagram



A-13) List of Servo to Pin Connections

Left Side

L1F - pin PB5
L2F - pin PE4

L1M - pin PE5
L2M - pin PB4

L1B - pin PA6
L2B - pin PA7

Right Side

R1F - pin PF2
R2F - pin PF3

R1M - pin PF0
R2M - pin PC5

R1B - pin PB7
R2B - pin PB6

A-14) TM4C123GH6PM Features

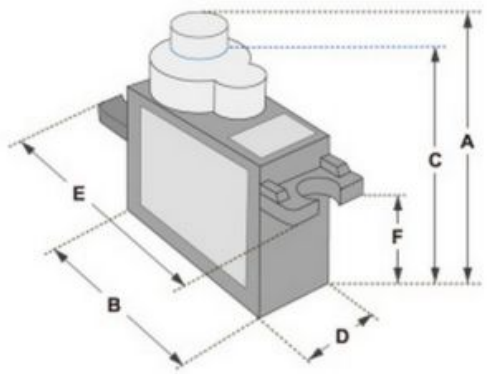
Feature	Description
Performance	
Core	ARM Cortex-M4F processor core
Performance	80-MHz operation; 100 DMIPS performance
Flash	256 KB single-cycle Flash memory
System SRAM	32 KB single-cycle SRAM
EEPROM	2KB of EEPROM
Internal ROM	Internal ROM loaded with TivaWare™ for C Series software
Security	
Communication Interfaces	
Universal Asynchronous Receivers/Transmitter (UART)	Eight UARTs
Synchronous Serial Interface (SSI)	Four SSI modules
Inter-Integrated Circuit (I^2C)	Four I^2C modules with four transmission speeds including high-speed mode
Controller Area Network (CAN)	Two CAN 2.0 A/B controllers
Universal Serial Bus (USB)	USB 2.0 OTG/Host/Device
System Integration	
Micro Direct Memory Access (µDMA)	ARM® PrimeCell® 32-channel configurable µDMA controller
General-Purpose Timer (GPTM)	Six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks
Watchdog Timer (WDT)	Two watchdog timers
Hibernation Module (HIB)	Low-power battery-backed Hibernation module
General-Purpose Input/Output (GPIO)	Six physical GPIO blocks
Advanced Motion Control	
Pulse Width Modulator (PWM)	Two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs.
Quadrature Encoder Interface (QEI)	Two QEI modules
Analog Support	
Analog-to-Digital Converter (ADC)	Two 12-bit ADC modules, each with a maximum sample rate of one million samples/second
Analog Comparator Controller	Two independent integrated analog comparators
Digital Comparator	16 digital comparators
JTAG and Serial Wire Debug (SWD)	One JTAG module with integrated ARM SWD
Package Information	
Package	64-pin LQFP
Operating Range (Ambient)	Industrial (-40°C to 85°C) temperature range Extended (-40°C to 105°C) temperature range

A-15) NodeMCU Specifications

Specifications of ESP-12E WiFi Module

Wireless Standard	IEEE 802.11 b/g/n
Frequency Range	2.412 - 2.484 GHz
Power Transmission	802.11b : +16 ± 2 dBm (at 11 Mbps) 802.11g : +14 ± 2 dBm (at 54 Mbps) 802.11n : +13 ± 2 dBM (at HT20, MCS7)
Receiving Sensitivity	802.11b : -93 dBm (at 11 Mbps, CCK) 802.11g : -85 dBm (at 54 Mbps, OFDM) 802.11n : -82 dBm (at HT20, MCS7)
Wireless Form	On-board PCB Antenna
IO Capability	UART, I2C, PWM, GPIO, 1 ADC
Electrical Characteristic	3.3 V Operated 15 mA output current per GPIO pin 12 - 200 mA working current Less than 200 uA standby current
Operating Temperature	-40 to +125 °C
Serial Transmission	110 - 921600 bps, TCP Client 5
Wireless Network Type	STA / AP / STA + AP
Security Type	WEP / WPA-PSK / WPA2-PSK
Encryption Type	WEP64 / WEP128 / TKIP / AES
Firmware Upgrade	Local Serial Port, OTA Remote Upgrade
Network Protocol	IPv4, TCP / UDP / FTP / HTTP
User Configuration	AT + Order Set, Web Android / iOS, Smart Link APP

A-16) SG-90 Specifications



Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.