Building Energy Managment Internet of Things

by

Reece Bachman and Jordan Ingram and Robert O'Malley
Advisor: Dr. Suruz Miah

Electrical and Computer Engineering Department
Caterpillar College of Engineering and Technology
Bradley University

Abstract

This paper presents a method for controlling multiple Internet of Things (IoT) devices. This work uses an open source software from Virginia Tech called Building Energy Management Open Source Software (BEMOSS) to connect multiple IoT devices to one server. This server is run on a linux laptop and manage a variety of IoT devices. This papers work allows the use of BEMOSS on a Raspberry Pi, a cheap commercially available microcontroller, to run commands through BEMOSS. Currently BEMOSS supports a number of commercial IoT devices and in this work, a new device, a DC motor, has been implemented to be run on BEMOSS. Lastly a heating, ventilating, and air conditioning (HVAC) control algorithm has been implemented to be later introduced onto BEMOSS.

Acknowledgements

We would like to thank our advisor, Dr. Suruz Miah, for the guidance and insight for this project. We would also like to thank Mr. Christopher Mattus for the aide in the construction process of our research project.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

As smart power grids become more prevalent and data on smart building energy usage becomes more available, the opportunity for energy efficiency increases drastically. Data collected from smart buildings allows control algorithms to dictate the workings of a building in a more efficient manner. We propose an expansion of the supported loads that can be controlled through BEMOSS by enabling control of a DC motor. Building energy management open source software (BEMOSS) is an internet of things (IoT) software that was developed at Virginia Tech under the Department of Energy funding. This proposed control of a DC motor through an IoT software presents the opportunity to close blinds, open barriers, and move objects throughout the building. The ability to open and close blinds building wide permits the ability to regulate the interior temperature of a building with a considerably lower power consumption. For instance, closing blinds during a hot day will naturally cool a room while opening blinds during a cold day will naturally heat the room. This added ability to control the interior environment will reduce the energy consumed by a heating ventilation air conditioning (HVAC) system.

1.1 Background Study

Authors in [?] proposed a energy management system based around Zigbees and PLCs. Han's system would place a energy measurement and communication unit in each outlet and light in the consumer's home. The energy usage will be measured and the data collected will be sent to a home server run on a Zigbee. These Zigbees will analyze the data and give feedback to the user on how to better manage their energy usage. Renewable energy will be connected to a PLC to allow the use of renewables as they need to converted to AC. The home server on the Zigbee will predict the amount of energy that will be obtained by renewables by accessing weather data and automatically adjust the user's device schedule.

In [?] the authors once again proposed sensors in all outlets and lights to measure the energy usage in homes. Renewable energy sources, like solar panels and wind energy, are connected to an inverter and battery system, to allow the storage of excess power. Collotta connects the internet of things devices using Bluetooth rather than WiFi, due to the lower power consumption. The system checks the current time and cross checks it with the peak time for energy consumption by the user's energy provider. If it is during peak times, the system checks the battery system for excess stored energy. When the user is trying to use energy during a peak time and without a battery charge the system warns the user, but allows the user to ignore these warnings.

In [?] the energy system uses two model predictive controllers (MPC), one for the building's HVAC system and one for the system's battery power flow. The building's energy management system predicts the temperature of each room separately using sensors and a Kalman filter for robustness. The battery system put in place by Mantovani can run on two modes, one to minimize energy cost and another to minimize power flow. The building is also c-onnected to wind turbines and photo-voltaic cells and predicts the energy produced using a simplified model.

The authors in [?] propose a Internet of Energy network rather than an IoT network. An IoE combines IoT and smart grid technology using four key components: Energy Router, Storage System, renewable sources, and plug-and-play appliances. The IoE allows for an easier way to produce a net zero energy building, that produces as much or more energy than it consumes. The energy router consists of a solid-state-transformer, grid control, and communication meant for data management. The storage system like batteries, reduce the stress on the power grid and lower voltage fluctuation. Renewable sources reduce carbon emmisions, however they reduce harmonics that need to be handled with additional hardware. Lastly the plug-and-play appliances are the devices that the end-user uses in a home.

In [?] the system heavily interfaces with the users' smartphones as a way to monitor the building occupants. Since smartphones almost all have a way to track GPS, the system tracks the users' location and send it to the building's server. The building's server is broken up into a number of subservers to handle an individual room's needs. By tracking location the building can do such things like pre-heating or pre-cooling a room before the user is even in the building. All the subservers are connected to the main server which is connected to cloud storage which is used for hosting the large amount of data and handling the intense computations.

1.2 Project Statement

Our project had three main goals. The first goal was to implement a new device not currently supported by BEMOSS on BEMOSS. The second goal was to run BEMOSS on a single board computer, such as a raspberry pi. The last goal is to develop a control algorithm to reduce energy consumption and implement the algorithm on BEMOSS.

Chapter 2

Overview

Below is the high level topology of our proposed BEMOSS system. The top right box titled "BEMOSS control center" represents the server and control module responsible for creating and executing commands to the supported IoT devices. BEMOSS runs a server to host and relay commands to the connected IoT devices. The control of these devices can be done on the same machine or by connecting to the server's web page on a separate machine. The control center communicates to our proposed "Wireless sensor central node" via a TCP/IP network. This node's purpose is to relay the WiFi commands from the control center to radio frequency commands to be received by the new DC motor IoT device. Here, we employ a Raspberry Pi 3B. Our Raspbery PI 3B connects to the same WiFi network as the control center. Commands from the WiFi network call scripts to relay communications over a Zigbee network. Commands from the control center to the already supported IoT devices are delivered through the TCP/IP network directly without passing through the central node.

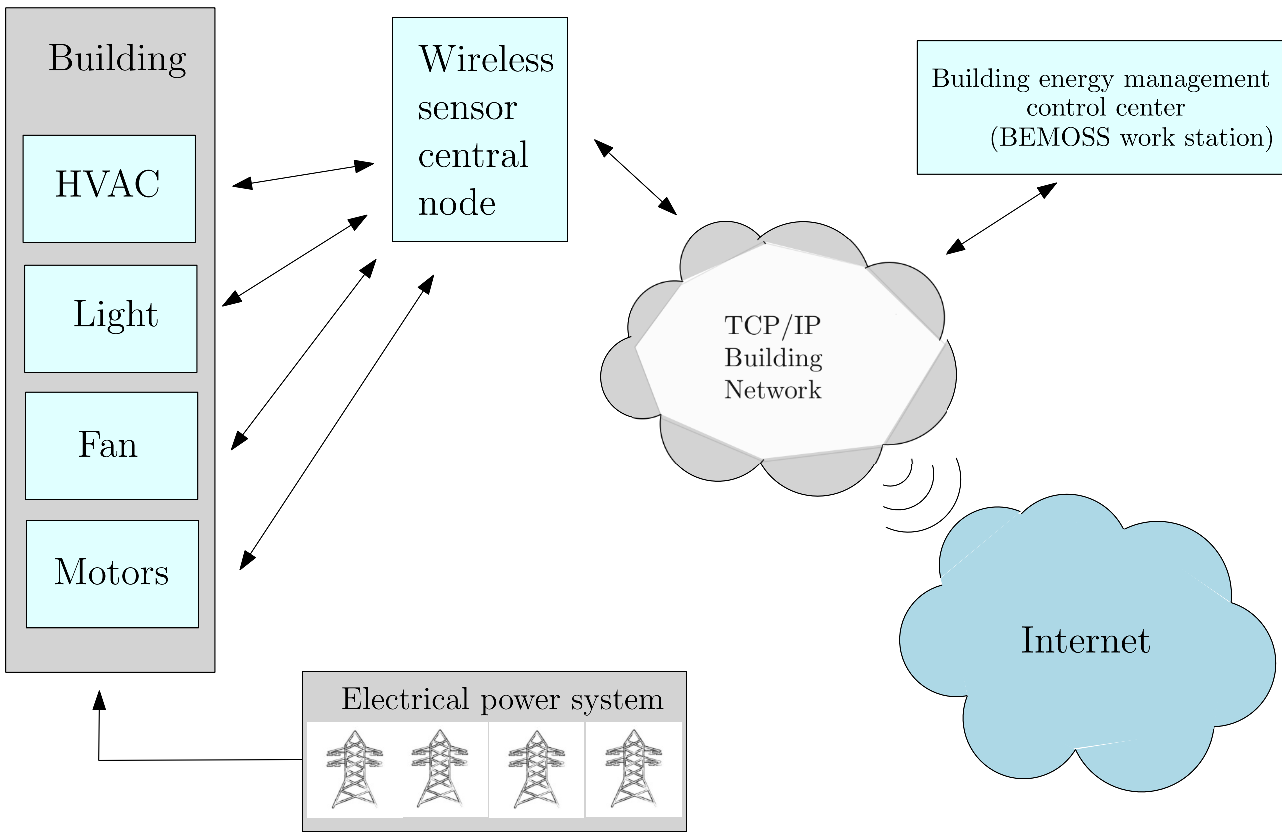

Building
HVAC
Light
Fan
Motors
Wireless sensor central node
Building energy management control center (BEMOSS work station)
TCP/IP Building Network
Internet
Electrical power system

Figure 2.1: High level BEMOSS structure

Chapter 3

Control Algorithm

To start the group wanted to create an algorithm that would monitor all devices on BEMOSS and use machine learning to optimize the consumer's behavior. To begin however we developed models for an HVAC system that could currently be controlled on BEMOSS but has no built in energy management support and a model of a motor that we developed to be controlled on BEMOSS.

3.1 System Modeling

To develop our models we used Matlab and Simulink and created block diagram models for a HVAC system as well as a motor controlled using a PWM signal and H-bridge. To create realistic models we used a Simulink library, Simscape, that has built in electromechanical blocks that we can adjust the parameters of to better model our real world system.

3.1.1 Motor

The model of the motor was based on the initial design constraints of our hardware. As can be noted the model is controlled witha PWM signal and H-bridge that can be used to run the motor in clockwise and counterclockwise directions. A current sensor is used to read the current output so that the power calculation can be done later in the model. A rotational sensor reads the rotation of the motor in rotations per minute. The DC motor is based on the Pittman 24V DC motor we are currently controlling through BEMOSS in the lab. To do this a DC motor block from simscape is used and the specifications are defined as they appear in the Pittman datasheet. This allows an accurate reading, however it should be noted that the blocks used in Simulink are ideal blocks with no losses to friction or transmission losses.

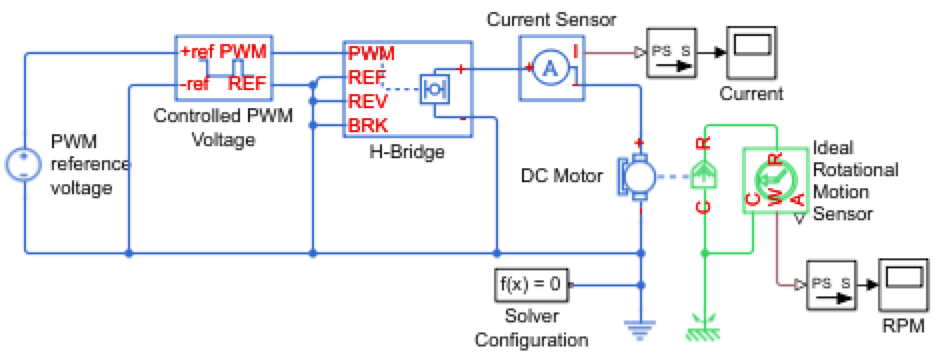


Figure 3.1: Motor Model in Simulink

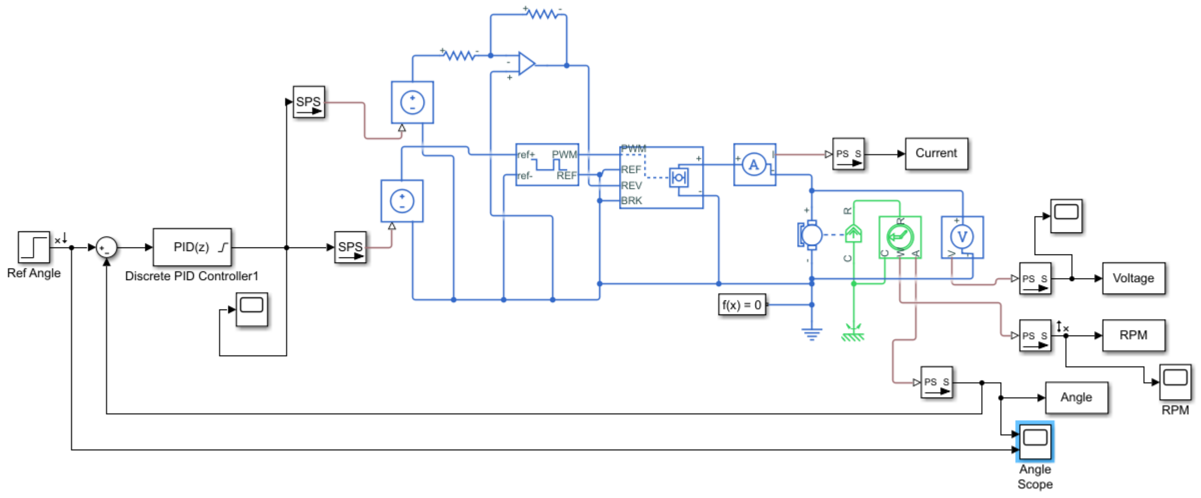


Figure 3.2: Motor Model With Position Control

3.2 Motor Control

As stated earlier a usage for the motor device being developed for this project is for opening and closing a curtain for home automation. As this is the case a control algorithm was developed using the simulink model in the previous section to control the position of the motor. The input variables for the system are motor applied voltage and the output variables are motor position.
An extension of the simulink model was needed to be added to control the motor position. The motor speed measurement block in simscape can also output motor angular position which allows the motor angle to be read. This is feedback to the start of the model and the difference between the reference angle that is desired and the current position. This creates the error signal that is run through a classic controller, the PID, or proportional integral and derivative controller. For this controller the user defines three gains to control the system that can be found in simulation through trial and error.

In figure 3.2 it can be seen that a discrete PID controller is used to power the PWM

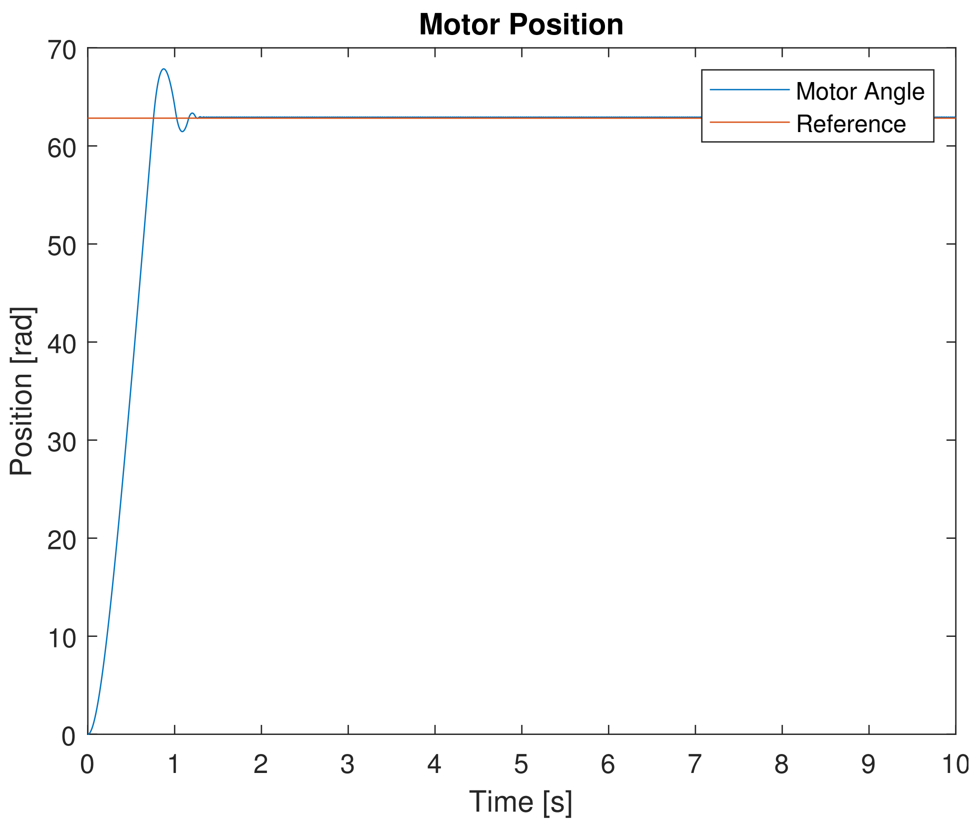


Figure 3.3: Motor Position Step Response

signal being fed into the motor. The gains of the controller are: $K_P = 2, K_I = 0$, and $K_D = 0.025$. A discrete PID control is more practical for embedded implementation for future works.
In figure 3.3 the step response to the motor is recorded with the motor rotating ten times. The rise time for the system was found to be 0.875 seconds with an 8 percent overshoot. The settling time is 1.25 seconds and the system converges onto the reference angle.

3.2.1 HVAC Model

In this HVAC House Model in Figure 3.2, the thermal properties of a one room home is modeled. An on-off heating control is used to control the temperature of the house. A thermostat block is used to determine if the current temperature in the house model is below the reference signal and using this basic check switches the heating system on or off. The house model itself exchanges heat between three components: air to window, air to wall, and air to roof. This is a simple modeling that is used to show the typical behavior of a house, even if the house itself is not a real world object. The outside temperature is a fluctuating sine wave to determine if the simple on off control will handle changes in temperature.

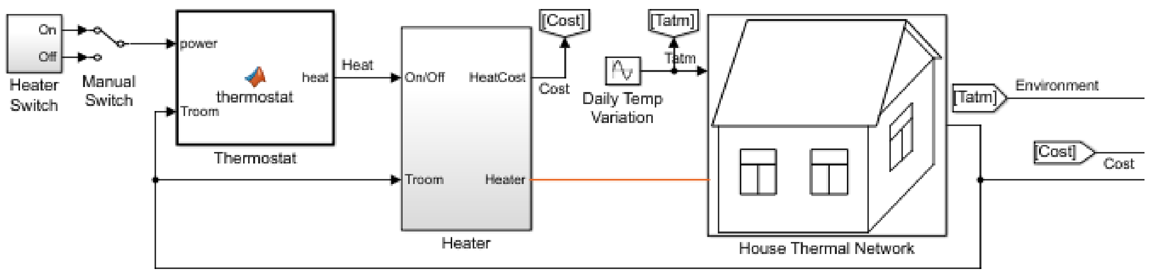


Figure 3.4: HVAC House Model

$$\boldsymbol{A} = diag\left[\begin{bmatrix}0&1&0\\0&0&1\\0&0&0\end{bmatrix}\begin{bmatrix}0&1&0\\0&0&1\\0&0&0\end{bmatrix}\begin{bmatrix}0&1&0\\0&0&1\\0&0&0\end{bmatrix}\right];$$

$$\boldsymbol{B} = diag\left[\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}\right];$$

$$\text{and } \mathbf{C} = \text{diag}[[1 \quad 0 \quad 0] \quad [1 \quad 0 \quad 0] \quad [1 \quad 0 \quad 0]].$$

Figure 3.5: System Matrices

3.2.2 State-Space Representation of HVAC System

To better design a more realistic model the team looked at the works of [?]. The design put forth by the team in [?] is designed using a State-Space Representation. The typical control problem is defined as

$$\dot{x} = Ax + Bu \tag{3.1}$$

The system used in [?] is a conventional HVAC system used in many buildings that controls temperature and humidity. In addition the system also measures and controls CO_2.

3.3 Control of HVAC Model

To begin a conventional HVAC system that controls only temperature and humidity will be defined and later in the report, built upon to include CO_2 monitoring. The modeling of the system begins by defining the system equations based on the control variables which are: volumetric air flow rate, the chilled water flow rate, and the outdoor air flow rate. Using these control inputs with outputs of: temperature, humidity ratio, and CO2 concentration of the thermal space, the system of equations is as follows.

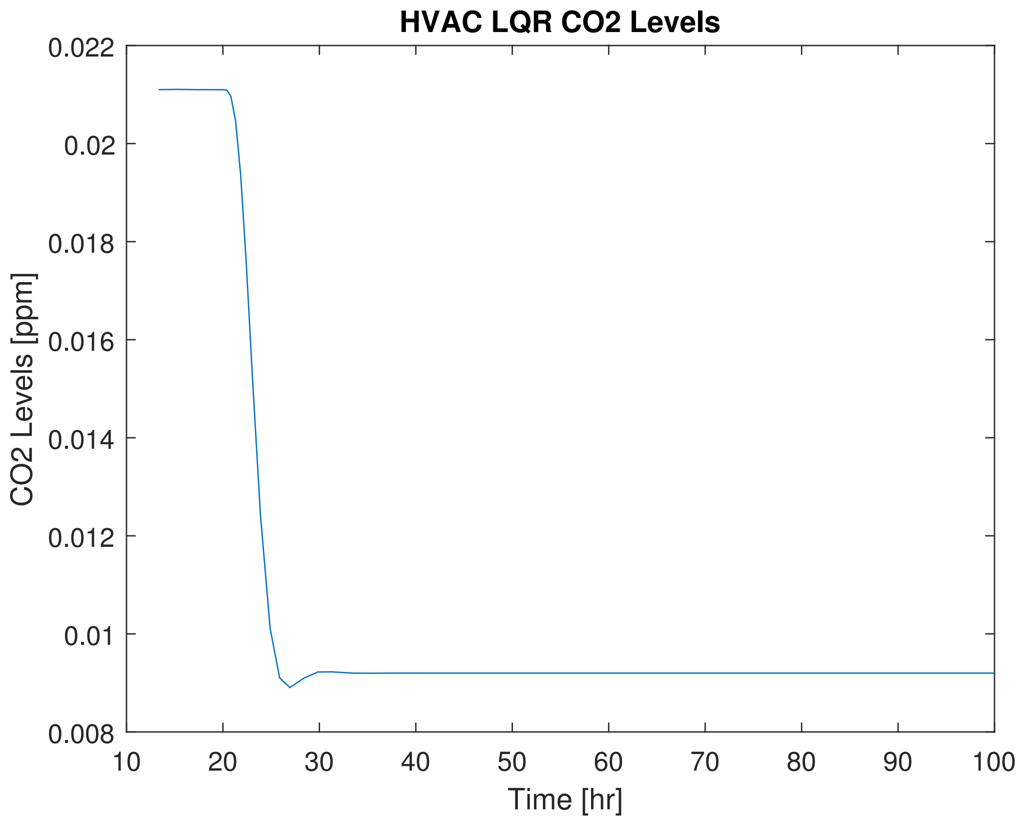


Figure 3.6: CO2 HVAC Response

Using the matrices from figure 3.3 the system can be converted to its error dynamical model using the following method.

$$\bar{A} = \begin{bmatrix} A & 0_{nxp} \\ -C & 0_{pxp} \end{bmatrix} \bar{B} = \begin{bmatrix} B \\ 0_{pxm} \end{bmatrix} \tag{3.2}$$

In this model the LQR control can easily be created by using MATLAB's built in function for LQR generation. The n in this case for the above equations comes from the system order, in this case n=9. The p is from output dimensions, in this case p=3. The m is from input dimenisons, in this case m=3.
To create the optimal PI controller using the LQR function, the user also needs to define the Q and R matrices which create the weights for the state variables and control variables, respectfully. In this case Q and R were found through trail and error until a output was found that was robust and converged in a reasonable time.

3.4 HVAC Control Results

From the figures 3.6, 3.7, and 3.8, it can be seen that the proposed LQR control for the system is stable.

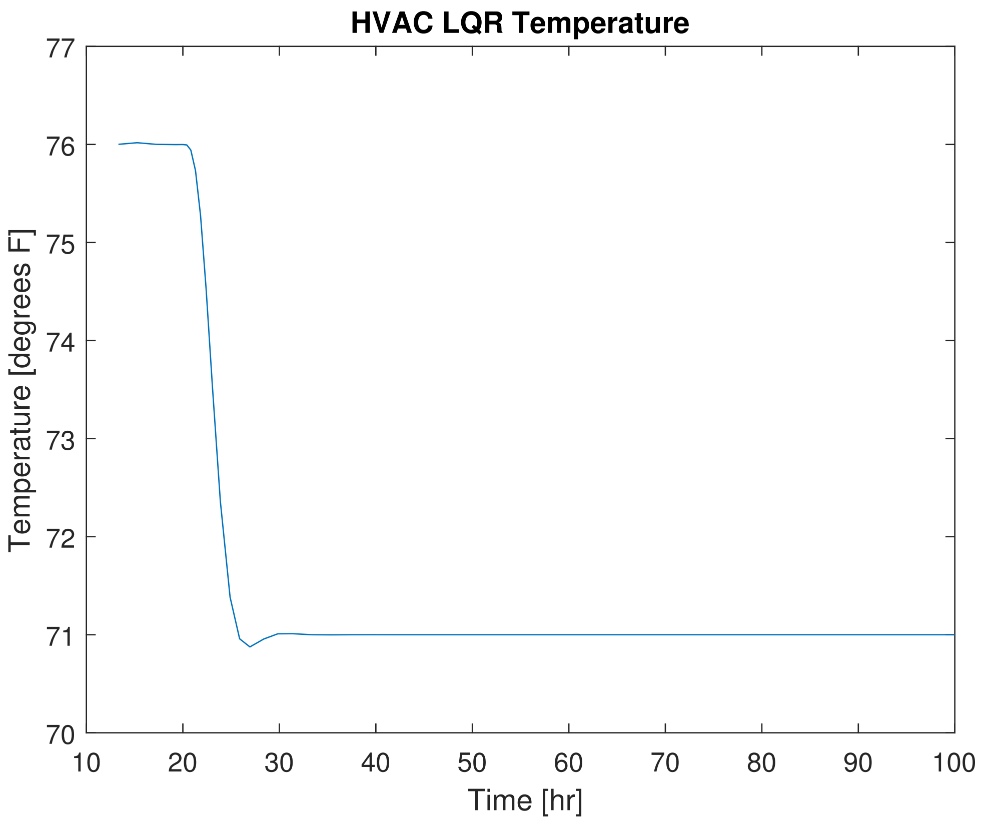


Figure 3.7: Temperature HVAC Response

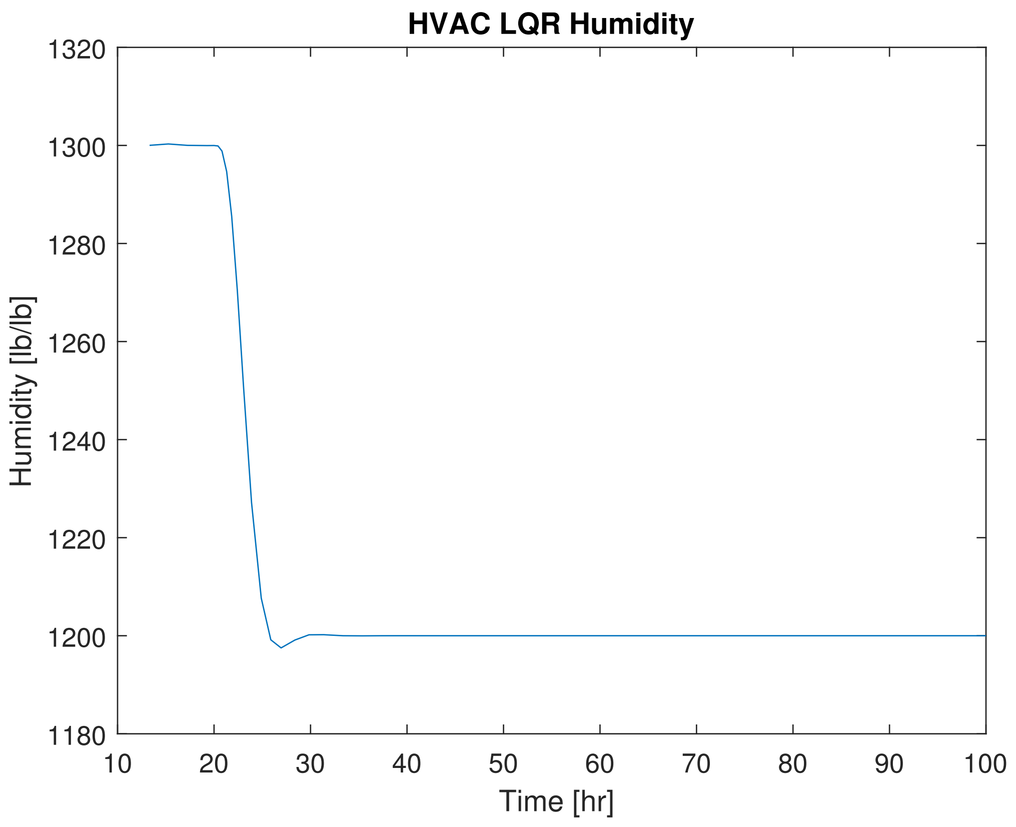


Figure 3.8: Humidity HVAC Response

Chapter 4

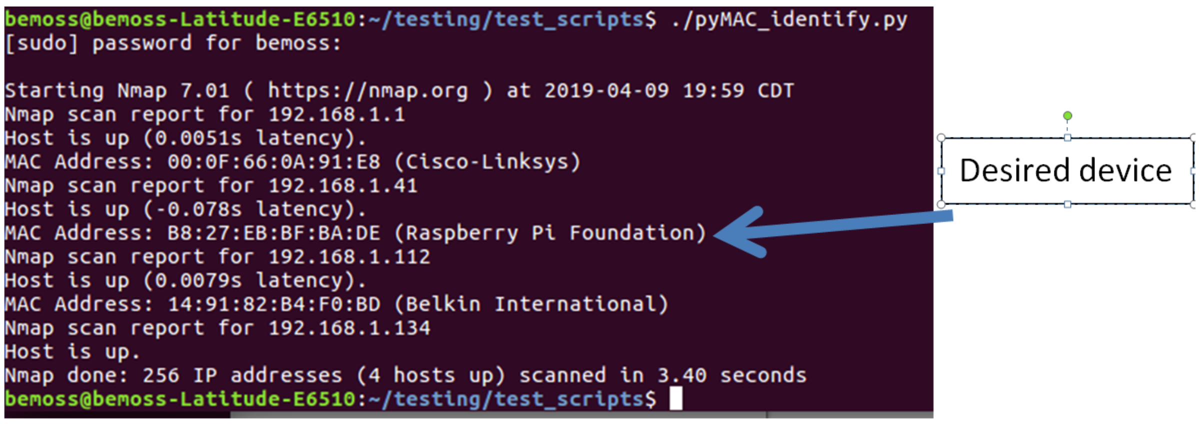
Implementing BEMOSS

4.1 Overview

Adding a new device into the supported devices of BEMOSS requires 4 segments of code: Ability to find the device, ability to control the device, editing the HTML file for added interface for the user, and editing the Django python files.

4.2 Discovery

The most critical part of adding new implementation to BEMOSS is being able to find the device that will be used. The way this is done is by finding all devices that are on the network. This in turn means the device must be able to connect to the network in order to be discovered by BEMOSS. Once all of the devices on the network have been discovered, the user must find the interesting device using the data that was given, as shown in the figure below.



```
bemoss@bemoss-Latitude-E6510:~/testing/test_scripts$ ./pyMAC_identify.py
[sudo] password for bemoss:

Starting Nmap 7.01 ( https://nmap.org ) at 2019-04-09 19:59 CDT
Nmap scan report for 192.168.1.1
Host is up (0.0051s latency).
MAC Address: 00:0F:66:0A:91:E8 (Cisco-Linksys)
Nmap scan report for 192.168.1.41
Host is up (-0.078s latency).
MAC Address: B8:27:EB:BF:BA:DE (Raspberry Pi Foundation)
Nmap scan report for 192.168.1.112
Host is up (0.0079s latency).
MAC Address: 14:91:82:B4:F0:BD (Belkin International)
Nmap scan report for 192.168.1.134
Host is up.
Nmap done: 256 IP addresses (4 hosts up) scanned in 3.40 seconds
bemoss@bemoss-Latitude-E6510:~/testing/test_scripts$
```

Figure 4.1: Caption

The data that we receive has the IP address as well as the MAC address of every device on the network. Using the MAC address, we can determine the important device since different manufacturers are provided different MAC address beginnings. Since we got the IP address, we are able to communicate with the device via internet. This will require a separate script that controls the motor. In order to enable the use of ssh, which is the way we were communicating with the motor, the following commands must be executed in the raspberry pi.

```
sudo systemctl enable ssh
sudo systemctl start ssh
```

From here the user is able to run a script on the raspberry pi from the main computer

4.3 Control

Controlling a device is a different task for each device that needs to be controlled so this section will talk specifically about the motor. There needs to be a python script on the raspberry pi that is able to control the motor however we want. Once that script is in place, we use the previously discovered ip to connect to the device and run the created script. This will control the motor how we want.

4.4 HTML

Editing the HTML file will allow for the use of Django web server that BEMOSS is currently employing. Finding the directory that the HTML file is under is a bit of a task considering there is different HTML files for each area. This project will be discussing only adding the HTML files for discovery. The directory in which the HTML file is located is the following

```
BEMOSS3.5\Web_Server\webapps\discovery\templates\discovery
```

This directory path can be changed anywhere after webapps to find the different HTML files located in BEMOSS.

4.5 Django

The final task to add functionality to BEMOSS is to add the scripts into a known directory and edit the correct Django python scripts, located in the directory below

```
BEMOSS3.5\Web_Server\webapps\discovery
```

In views.py, a new function must be defined which will execute the commands to control the motor and then provide a response to BEMOSS with relevant information depending on the outcome of the process. This python function will be called by going into a url that has been specified in the next script.

urls.py defines the function that is called when a request is made to a specific url. The function defined in views.py is called when clicked on a button that was defined within the HTML file.

Chapter 5

Interfacing New IoT Device

5.1 Overview

The new IoT device has 3 main parts: DC motor/ DC motor control circuit, Radio Frequency (RF) communication modules, and the central node. The central node receives commands from the BEMOSS control center over WiFi communications. In this central node, the processes, algorithms, and control methods of the new device are implemented and executed. The central node relays commands to the remote DC motor through the RF XBEE communication modules. The control circuit is kept cheap but utilizing simple pin toggling of the XBEE module to control the motor direction. Lacking on-board logic allows the system to expand without significantly raising the system price or part requirements. The DC motor control circuit simply reads the remote toggling of the XBEE General Purpose Input Output (GPIO) pins which toggle the H-Bridge pins, thus rotating the DC motor in the desired position.

5.2 Central Node

The central node is the command center of the newly developed IoT device. The central node utilized in this project is a Raspberry Pi 3. Any single-board computer capable of serial and WiFi communication will suffice for this role. As stated, the control algorithms for all wireless transmissions and feedback processes happen at the central node. Python scripts saved on this single-board computer can be called by WiFi commands. The python scripts in the project setup the communication ports, call XBEE transmission libraries, and send AT-commands through the serial port to command XBEE transmission.

5.3 XBEE Radio Frequency Modules

XBEE RF modules provide a robust and simple method of wireless communication and control. In an XBEE network, a command/ master module transmits commands to numer-

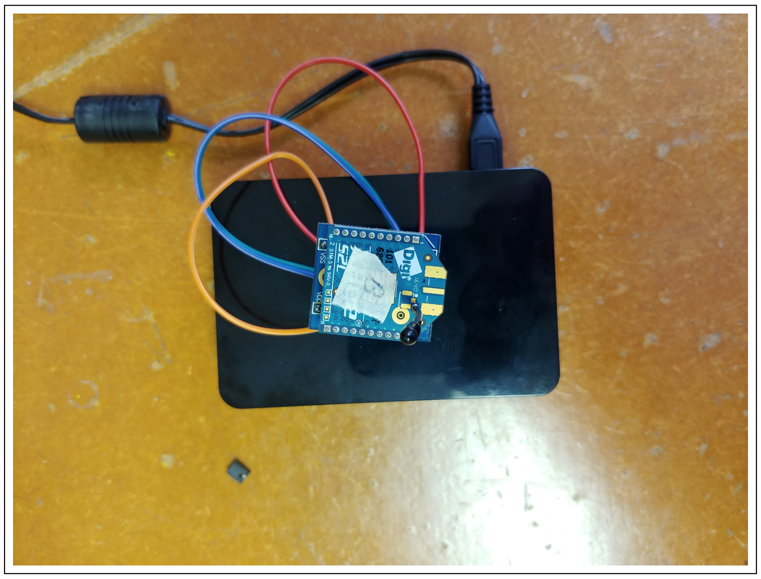


Figure 5.1: Central Node

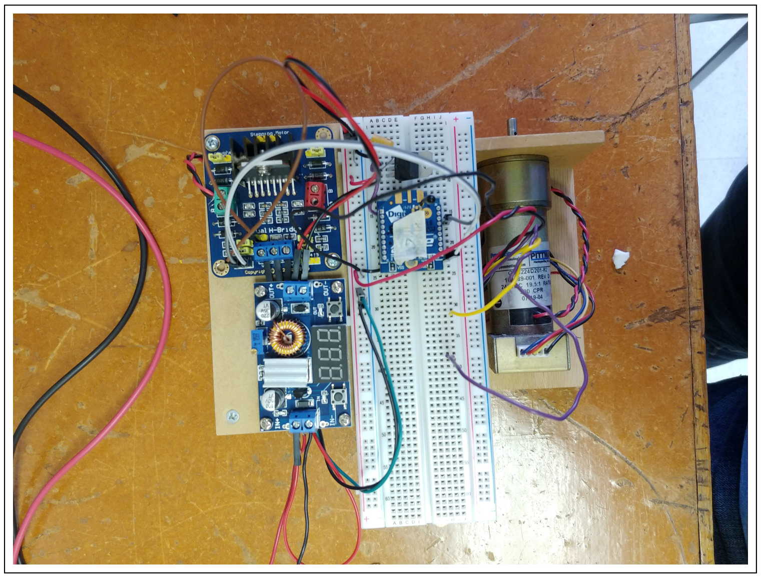


Figure 5.2: Remote DC Motor

ous slave XBEE modules. The XBEE modules can be initialized through many methods to complete a wide array of tasks. In our case, we initialized our modules through the XCTU software to toggle GPIO pins upon command from a user or computer. These commands can be transmitted either from the XCTU software interface or from serial transmission from a computer. Each XBEE has an address that can be used to communicate with specific XBEE modules at a time. After initializing and dictating master /slave XBEE relationships, the master XBEE was moved to be controlled by the central node. The central node transmits serial commands to the master XBEE module from it's TX GPIO pin.

5.4 DC Motor and Control Circuit

The DC motor control circuit consists of a buck/boost converter, a slave XBEE RF module, a H-Bridge, and a DC motor with an attached optical encoder. The buck/boost converter allows for a single power supply input of 5-24 volts to the unit. The supply of 5-24 volts is supplied directly to the motor while providing power to the buck/boost converter to supply the proper logic voltage to the control circuit. The power from the buck/boost converter is either bucked down or boosted up to a steady 5 volts. The H-Bridge requires a 5 volt logic to operate, while the slave XBEE module requires 3.3 volts. 3.3 volts is supplied to the slave XBEE module from the 5 volt on-board logic through a voltage regulator.

DC motors are simple as they only require 2 wires to control their speed and direction. The DC voltage applied to the positive and negative wires dictates the speed of the motor shaft rotation while the polarity of the applied voltage dictates the rotation direction. A motor driver is employed to switch polarity of voltage applied to these 2 wires. H-Bridges accept an input of motor supply voltage, pulse-width-modulation (PWM), and 2 pins to toggle motor rotation direction. The motor's leads are attached to the output of the H-Bridge. Our H-Bridge, a L298N, uses 2 sets of 2 transistors to switch the flow of power through the motor leads depending on the status it's input 1 and 2 pins. Each pin dictates the rotation direction. Both pins set to low results in no rotation. Pin 1 set to high while pin 2 is low results in clockwise rotation. Pin 2 set to high while pin 1 is low results in counterclockwise rotation. A short circuit will result if both pin 1 and pin 2 are set to high. PWM control can be supplied from our XBEE slave module's GPIO pins to the H-Bridge to control the speed of the motor rotation. PWM commands rapidly switch the power supply to the motor on and off resulting in a varied voltage supply to the motor. As stated, varied voltages to a DC motor results in variable speeds.

5.5 Central Node Software

place snippets of python codes here with functionality explanations

5.6 Search and Control Software

place snippets of Bash and python codes here with functionality explanations. Also place flow of data diagram here

5.7 Future Work

A motor position feedback system shall be implemented to allow for the precise positional control of the DC motor. With this precise control, the central node can command the motor to turn a direction while reading it's position from it's starting point. This ability allows a continuously monitored task to initiate the DC motor to turn and, when a desired position is achieved, cease turning .

The feedback control of the remote DC motor and control circuit is allowed to remain cheap through the use of the XBEE modules. The Pittman DC motor that is employed in this project has an optical encoder reading the rotation of it's shaft. As the motor shaft rotates, the optical encoder outputs a square wave as light passes through holes on a disk attached to the motor shaft. Through the count of high pulses from the encoder multiplied by the circumference of drive attached to the motor shaft, the distance traveled by the motor is calculated. As said, the remote DC motor control circuit has no logic on board, but the high and low pulses from the optical encoder can feed into the slave XBEE module on the motor and be initialized to relay high and low pusles to it's master. The relay of high and low pulses from the optical encoder to the slave XBEE module must pass through a logic convert as the optical encoder operates on 5 volt logic while the XBEE module opperates on 3.3 volt logic. The processing of the high and low toggles on the remote XBEE module will be received and processed on the single board computer in the central node. This process allows for numerous cheap motor control circuit to exist with a single processing unit to connect to a WiFi network.

Chapter 6

Software Set Up Appendix

6.1 Step 1: Installing Linux

A good explanation of this is written on the tutorials at

`https://tutorials.ubuntu.com/tutorial/tutorial-install-ubuntu-desktop-1604`

but will also be explained through this appendix BEMOSS currently only only supports one distribution of Linux, Ubuntu 16.04. This is not an issue since this is a free distribution. To install Ubuntu 16.04, we must first download it from the website onto a USB flash drive

`http://releases.ubuntu.com/16.04/`

Most computers have 64 bit processors but check the processor before downloading. After downloading to the USB drive, boot up the computer from this flash drive. This is often done by entering the BIOS upon start up unless no image has been written to the computer. To enter the BIOS, a prompt will often appear to click either the del key or f12 depending on the computer. Once entered, there are different options, one of which will be to choose boot device. After choosing the USB flash drive, the user will be presented with a screen to either try Ubuntu or to install Ubuntu.Click install Ubuntu and then follow the steps provided. If the machine being used to install Ubuntu is being used only for Ubuntu click Erase disk and install Ubuntu but if the machine also wishes to run another operating system it is suggested to do further research. Complete the following prompts and restart once Ubuntu is fully installed.

6.2 Step 2: Installing Software

This project requires a few different types of software for different functionalists within the distribution of Linux that we will be using. Ubuntu 16.04 commonly has everything except BEMOSS but it is important to make sure it is all there.

- BEMOSS
- git
- gedit
- Python
- bash

To check for installed software, open up a command terminal by right clicking the desktop and click command terminal. type in the command ”dpkg –get-selections” and the terminal will give the user a lot of information. This information is listing the installed packages on the computer so look for git, gedit, python, and bash. if any of these are not installed, type ”sudo apt-get install ¡package¿” where ¡package¿ is the name of the package not installed, git for example. Typing sudo will require the user to input the password. After git, gedit, python, and bash are found when the command is typed, move on to the next step

6.3 Step 3: Installing BEMOSS

The BEMOSS project can be downloaded off the website github, which is why we needed the git package. typing in the command

```
git clone -b master https://github.com/bemoss/BEMOSS3.5.git"
```

exactly as listed above will download BEMOSS into the current directory. After it has been completely downloaded, the user will install BEMOSS by navigate to the GUI directory within the BEMOSS folder that was created. To do so, the user will use the cd command to enter a directory, BEMOSS in the first case and then again to enter the GUI directory. The user will then type the following command in the terminal exactly as listed

```
./startBEMOSS_GUI.sh
```

This will launch the BEMOSS GUI where the user will then click run BEMOSS. The program will prompt the user if they would like to install BEMOSS as it has not yet been installed. After clicking yes, BEMOSS will install onto the machine with a few prompts at the end if successfully installed. The first is the user name which must be admin currently. Everything after user name is up to the users choice but should be remembered.

A few things still need to be changed in order to get BEMOSS to work. On the BEMOSS GUI, the user should click advanced setting and click database configuration files. 2 files open up and both need to be edited. the first one to edit, postgresql.conf, needs to change line 59

```
listen addresses = 'localhost' \# what IP address(es) to listen on;
```

```
listen addresses = '*' \# what IP address(es) to listen on;
```

The next file that needs to be add is hba.conf, line 82.

```
from
host    all    all    127.0.0.1/32
to
host    all    all    127.0.0.1/32        md5
host    all    all    10.0.2.15/32        md5
```

where 10.0.2.15 is the IP address of the machine being used. To check the ip address, open up a command terminal and type ifconfig and look for the IP address

The final file that needs to be edited is multinode_data.json. Modify the line that has *"address": tcp://192.168.10.62:9000"* so that it reads *"address": tcp://10.0.2.15:9000* where the numbers that changed are your IP again. Make sure to save everything that you have been editing and then restart the computer. The restart will finalize the changes and once running bemoss again with the command and running BEMOSS, the server should be deployed properly

```
./startBEMOSS_GUI.sh
```

6.4 Step 4: Adding functionality

In order to add functionality to BEMOSS, the user must have a python script that executes the desired functionality. For the purpose of this tutorial, the python script controlling the motor will be used. Replace the previous views.py and urls.py in the following directory

```
BEMOSS3.5/Web_Server/webapps/discovery]]
```

the html file must be edited in order to run the program which will take place on whatever url is placed in the url file and is changed in the directory

```
BEMOSS3.5/Web_Server/webapps/discovery/templates/discovery
```

Chapter 7

Hardware Set Up Appendix

7.1 Step 0: Parts used in this project

- Raspberry Pi 3 Model B
- L298N Motor Driver
- 24V Pittman DC motor with attached optical encoder
- 2 XBEE S2C radio frequency modules
- Buck/Boost Converter
- 2 XBEE USB explorers
- 2 Micro USB cables
- 3.3V regulator
- 0.1 uF Capacitor
- Jumper Wires
- Power supply (4-24v)

7.2 Step 1: Libraries and Settings Utilized

A few libraries need to be installed on the Raspberry Pi 3 Model B to work this project. The libraries listed at this time may not be sufficient as the software and protocols used in this project are ever evolving. These libraries can be downloaded through the command line. We wrote our code in python, so pip install commands in the command line was our method of install. We had issues downloading the XBee libraries on our original Raspberry Pi 3, so we exchanged for a new one and the libraries installed fine.

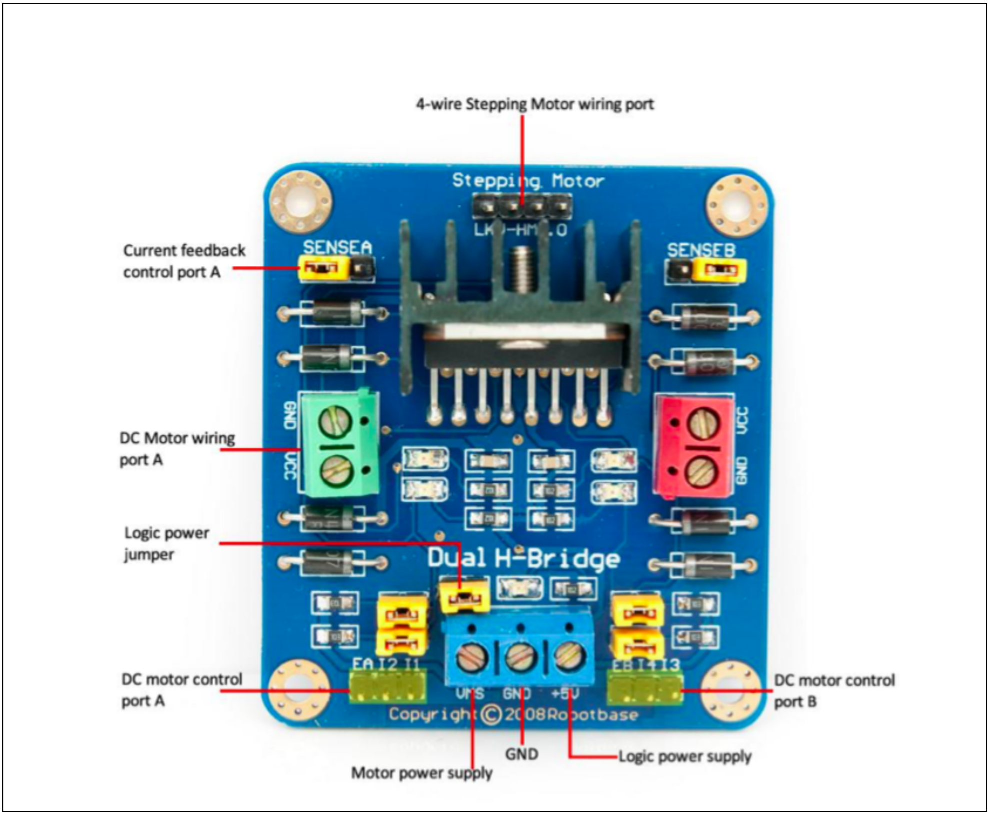


Figure 7.1: L298N Dual H Bridge Schematic.

- Enable Raspberry Pi General Purpose Input/ Output (GPIO) pins
- XBee libraries
- Enable serial port communication
- install EMACS (text editor to write python files in. Save Files as .py)

7.3 Step 2: Hardware Connections

- See below figures for XBEE and L298N diagrams, connections, and specifics
- Main power supply connected to the input of the Buck Boost
- L298N has 2 sets of 2 transistors to toggle motor direction. The pairs of transistors are controlled by their designated I1 and I2 input pins. Reference pas 40-46 for Pin toggle and motor rotation combinations
-

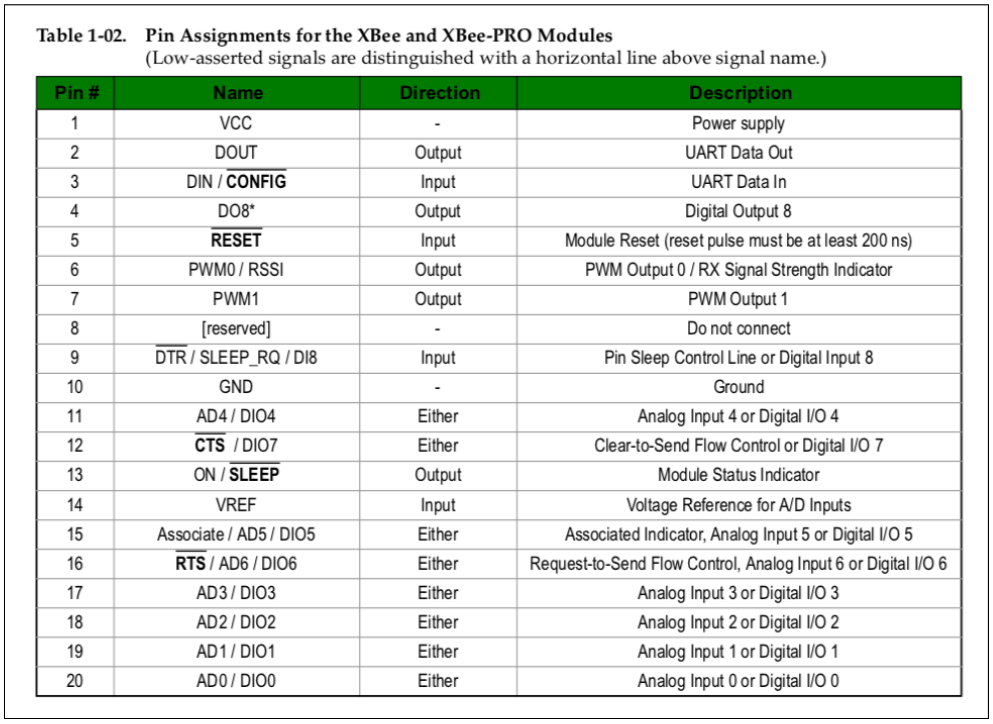


Table 1-02. Pin Assignments for the XBee and XBee-PRO Modules
(Low-asserted signals are distinguished with a horizontal line above signal name.)

Pin #	Name	Direction	Description
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / $\overline{\text{CONFIG}}$	Input	UART Data In
4	DO8*	Output	Digital Output 8
5	$\overline{\text{RESET}}$	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI	Output	PWM Output 0 / RX Signal Strength Indicator
7	PWM1	Output	PWM Output 1
8	[reserved]	-	Do not connect
9	$\overline{\text{DTR}}$ / SLEEP_RQ / DI8	Input	Pin Sleep Control Line or Digital Input 8
10	GND	-	Ground
11	AD4 / DIO4	Either	Analog Input 4 or Digital I/O 4
12	$\overline{\text{CTS}}$ / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / $\overline{\text{SLEEP}}$	Output	Module Status Indicator
14	VREF	Input	Voltage Reference for A/D Inputs
15	Associate / AD5 / DIO5	Either	Associated Indicator, Analog Input 5 or Digital I/O 5
16	$\overline{\text{RTS}}$ / AD6 / DIO6	Either	Request-to-Send Flow Control, Analog Input 6 or Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0	Either	Analog Input 0 or Digital I/O 0

Figure 7.2: XBee Pin Configuration

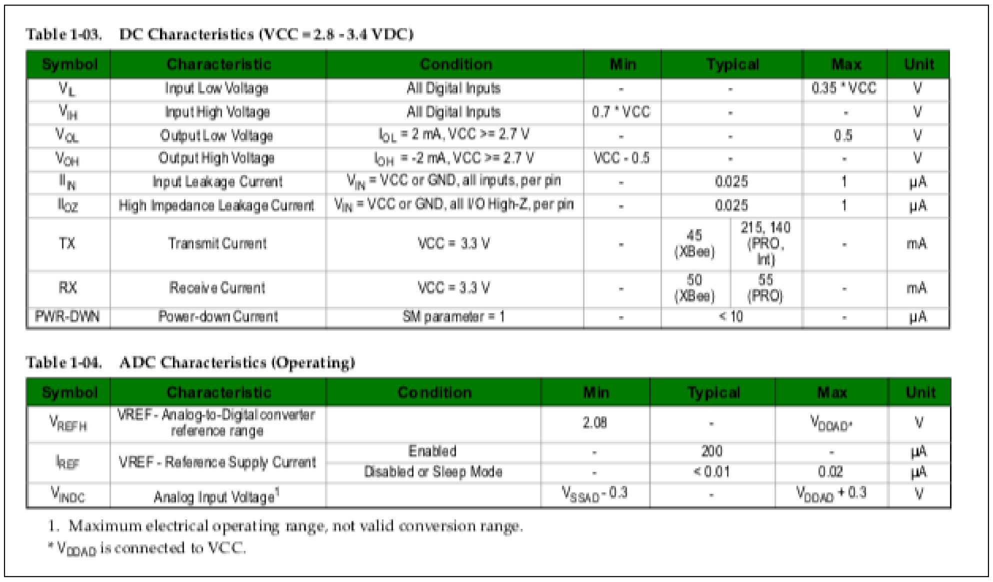


Table 1-03. DC Characteristics (VCC = 2.8 - 3.4 VDC)

Symbol	Characteristic	Condition	Min	Typical		Max	Unit
V_L	Input Low Voltage	All Digital Inputs	-	-		0.35 * VCC	V
V_{IH}	Input High Voltage	All Digital Inputs	0.7 * VCC	-		-	V
V_{OL}	Output Low Voltage	I_{OL} = 2 mA, VCC >= 2.7 V	-	-		0.5	V
V_{OH}	Output High Voltage	I_{OH} = -2 mA, VCC >= 2.7 V	VCC - 0.5	-		-	V
II_{IN}	Input Leakage Current	V_{IN} = VCC or GND, all inputs, per pin	-	0.025		1	µA
II_{OZ}	High Impedance Leakage Current	V_{IN} = VCC or GND, all I/O High-Z, per pin	-	0.025		1	µA
TX	Transmit Current	VCC = 3.3 V	-	45 (XBee)	215, 140 (PRO, Int)	-	mA
RX	Receive Current	VCC = 3.3 V	-	50 (XBee)	55 (PRO)	-	mA
PWR-DWN	Power-down Current	SM parameter = 1	-	< 10		-	µA

Table 1-04. ADC Characteristics (Operating)

Symbol	Characteristic	Condition	Min	Typical	Max	Unit
V_{REFH}	VREF - Analog-to-Digital converter reference range		2.08	-	V_{DDAD}*	V
I_{REF}	VREF - Reference Supply Current	Enabled	-	200	-	µA
		Disabled or Sleep Mode	-	< 0.01	0.02	µA
V_{INDC}	Analog Input Voltage[1]		V_{SSAD} - 0.3	-	V_{DDAD} + 0.3	V

1. Maximum electrical operating range, not valid conversion range.
* V_{DDAD} is connected to VCC.

Figure 7.3: XBee Electric Data

7.4 Step 3: Configure XBEE Modules

- Download XCTU to laptop/computer. This is the main XBEE configuration and communication software
- review online tutorials of XBEE prior to changing settings
- Online tutorials thoroughly describe which parameters to set to fulfill your project.
- Thoroughly reading the lab notebook can result in avoiding time consuming errors and findings.

- Master XBEE Setup
- we used a USB to XBEE board to configure the XBEE on the XCTU software.
- PAN ID: 1234 (all modules in network must have same PAN ID
- DL Destination Address: 415B96BF
- NI Node Identifier: RPi
- SH: 13A200
- SL: 415B96A7
- DH:0
- DL:FFFF
- modules transmissions received by all modules

- Slave XBEE setup
- DL: FFFF
- NI Node Identifier: MotorDrive
- SH: 0013A200
- SL: 415B96BF

- SL: 41630E56
- Channel C
- PAN ID: 1234
- DH=DL=0

- MY=16 bit source address =0
- SH=13A200
- SL: 41630E56

If a module's DH is 0 and its DL is less than 0xFFFF (i.e. 16 bits), data transmitted by that module will be received by any module whose 16-bit address MY parameter equals DL

If DH is 0 and DL equals 0xFFFF, the module's transmissions will be received by all modules.

If DH is non-zero or DL is greater than 0xFFFF, the transmission will only be received by the module whose serial number equals the transmitting module's destination address (i.e. whose SH equals the transmitting module's DH and whose SL equals its DL).

Successful Frame transmit to set IO4 on remote xbee high:
7E 00 10 17 01 00 13 A2 00 41 63 0E 56 FF FE 02 44 34 05 AE

Successful Frame transmit to set IO4 on remote xbee low:
7E 00 10 17 01 00 13 A2 00 41 63 0E 56 FF FE 02 44 34 04 AF

Successful Frame transmit D3 High:
7E 00 10 17 01 00 13 A2 00 41 63 0E 56 FF FE 02 44 33 05 AF
Successful Frame Transmit D3 Low:
7E 00 10 17 01 00 13 A2 00 41 63 0E 56 FF FE 02 44 33 04 B0

7.5 Step 4: Software

- Write code in a text file and save as a .py file
- Execute python files by navigating the directory (cd (directory) to change directories and ls to list files in that directory) and typing the command "python XBEETEST.py" to execute the code to turn the motor both directions.

ODE FILES HERE

7.6 Step 4: Trouble Shooting

- Beware of credentials deleting themselves in the XCTU software upon rediscovery
- Ensure the proper port communication in software. Raspberry PI 3 Model A vs Model B employ different port protocols and naming conventions.

- Our Raspberry Pi produced endless errors due to improper library downloads. Reinstall the OS or replace with a new Raspberry Pi and download the same libraries and this problem should disappear.
- Some XBEE modules had trouble being discovered and communicating with each other. Ensure the settings are proper and replace with a new module if necessary.

Chapter 8

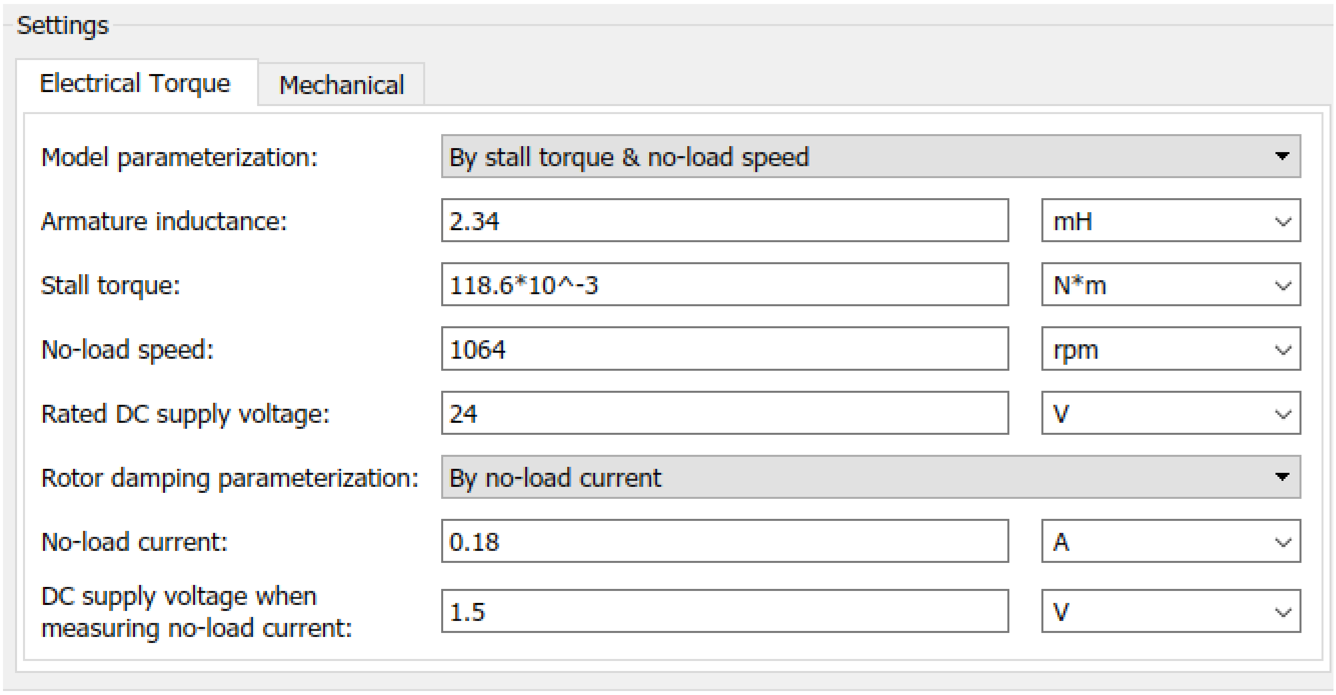
Control Appendix

8.1 Software used

- Matlab R2018a
- Simscape electrical 2018

8.2 Step 1 - Motor Model Creation

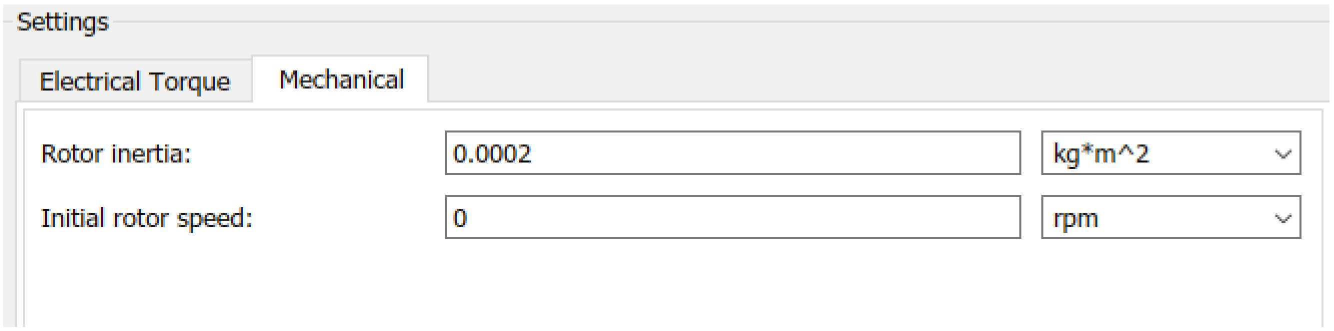
Begin by opening simulink. Create a new project and add a step block from the sources library, this block will be your reference angle block that will tell the motor what position to go to. Connect the step block to a sum block with a positive and negative sign. Create a discrete time PID block. Go into the parameters of the PID block and change the proportional gain to 2 and the derivative gain to 0.025. Go to the PID advanced tab and check the box to limit output. Set the upper saturation level to 1 and the lower saturation level to -1. Connect the output of the sum block to the input of the PID block. The output of the PID block will be connected to a Simulink-PS converter block. Create a controlled voltage source block and connect the PID output to the input of the voltage source with an arrow. Create a controlled PWM voltage block and change its parameters. Change the PWM frequency to 30 kHz. Go to the Input Scaling tab and change the input voltage for zero percent duty cycle to -2 V and change the input voltage for 100 percent duty cycle to -1V. Go to the Output Voltage tab and change the amplitude to 24V.
Add a H-bridge block and connect the PWM block's PWM signal to the PWM input of the H-bridge. Connect the REF output from the PWM to the REF and BRK inputs on the H-bridge. Go back to the output of the PID block and connect it to another Simulink-PS converter block. Connect this signal to another controlled voltage source. Create an inverting op amp configuration with unity gain connecting from this voltage source to the REV block on the H-bridge.
Add a DC motor block and connect the positive output of the H-bridge to the electrical side of the dc motor. Connect the negative terminal to ground along with the negative


Settings

Electrical Torque | Mechanical

Model parameterization:	By stall torque & no-load speed	
Armature inductance:	2.34	mH
Stall torque:	118.6*10^-3	N*m
No-load speed:	1064	rpm
Rated DC supply voltage:	24	V
Rotor damping parameterization:	By no-load current	
No-load current:	0.18	A
DC supply voltage when measuring no-load current:	1.5	V

Figure 8.1: Electrical Motor Specifications


Settings

Electrical Torque | Mechanical

Rotor inertia:	0.0002	kg*m^2
Initial rotor speed:	0	rpm

Figure 8.2: Mechanical Motor Specs

references for the voltage sources, PWM signal generator, and H-bridge. For the mechanical section of the motor, connect the C terminal to a mechanical rotational reference block. Connect the R terminal to an ideal rotational motion sensor. Connect the A output to a PS-Simulink block and connect that output to a simulink scope. Take the negative channel of the summer block from the beginning and connect it to the angle signal, completing the feedback loop.
Double click the motor block to open the configuration window. Set the parameters to the same as in figures 8.1 and 8.2.