ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE
Paper ID #37704

Leveraging ThingsBoard IoT Service for Remote Experimentation

Ahmet Can Sabuncu (Assistant Teaching Professor)

Dr. Sabuncu holds a Ph. D. in Aerospace Engineering from Old Dominion University. Dr. Sabuncu's professional interests spans from engineering education research, engineering laboratory education, history of science and engineering, thermo-fluids engineering, and microfluidic technology. Dr. Sabuncu is eager to discover next generation workforce skills and to educate next generation of engineers who will carry Industry 4.0 forward considering the needs of the global world.

Kerri Anne Thornton (Student / Lab Assistant & Teaching Assistant)

Leveraging ThingsBoard IoT Service for Remote Experimentation

Abstract

The goal of this study is to demonstrate the use of Internet of Things (IoT) technology to enable remote experimentation in mechanical engineering. In the 2021 ASEE Virtual Conference, we have shown one-way data transfer from an evaporative cooler setup, where students could remotely acquire data using Google Sheets. In this work, we describe a different scheme of IoT structure, where students could both acquire data and control equipment. We use an inexpensive (essentially no-cost for the community edition) opensource IoT platform, ThingsBoard. This software allows students to view dashboards that enable remote control of sensors and actuators, include livestreaming and live data as both plots or live numbers, and even allow students to download data for further analysis.

Our remote lab is an evaporative cooler designed to teach students about psychometrics, the thermodynamics of atmospheric air. The cooling effect is achieved by the phase transition of water in an evaporative pad. The experimental setup includes temperature and humidity sensors at the up-and downstream of the evaporative material. Students also can control a fan that blows air through the evaporative material. All sensors and actuators are connected to an Arduino board and a Raspberry Pi single-board computer, which interface to the ThingsBoard service for remote control and data acquisition. Finally, to facilitate a live view of the experimental setup in ThingsBoard, a Raspberry Pi also employs a camera that provides a live video feed of the entire setup while it is at work. We offer this exercise to students in a junior-level Engineering Experimentation course in a technological university. We report our findings in terms of students' perception of the experiment and student success in meeting the learning objectives. We also report our experience in having multiple users use a single remote experimentation

setup. Ultimately, we hope to match the trend of increasingly prevalent online learning with a low-cost solution where students have “almost” hands-on experimentation while also gaining invaluable skills in the world of IoT.

Introduction

Internet of Things (IoT) has become prominent in our lives with the Industry v4.0 revolution. IoT has also entered the engineering education space, partly because of the challenges that COVID-19 has brought. IoT technology enables students to actively participate in a ‘real’ experiment while being remote. IoT technology will likely stay in the engineering education space even after the COVID-19 pandemic is over as it offers a simple and low-cost way to enable remote experimentation. Hands-on activities in traditional laboratories have been offered in terms of real-time remote laboratories in mechanical engineering. For instance, at the Institute of Forming Technology and Lightweight Materials at TU Dortmund students experiment with a remote-controlled robot arm that handles specimens for cupping, tensile, and compression testing [1]. A remotely operated industrial-scale Flowloop test was developed for teaching multi-phase fluid mechanics at the Clausthal University of Technology [2]. In another study, students program control algorithms for operating mobile remote-controlled robots [3]. A pilot study reports the use of IoT for remote control and monitoring of a 3D printer as an academic exercise [4]. While the infrastructure for creating remote-controlled experiments with IoT is at a good maturity level, there are a very limited number of case studies, especially in the mechanical engineering area, available for educators to adopt.

In a recent study, we have demonstrated an evaporative cooler experiment that is remotely accessible by IoT [5]. In this study, students were able to observe and acquire data from an evaporative cooler in real-time. We enable remote control of the evaporative cooler in the current study. For this purpose, we used an IoT platform, ThingsBoard, to create a dashboard that contains device control and data display widgets. We test the current approach in an engineering experimentation course in a medium-size technological university and report the results here.

Description of the Experiment

Physical Setup

We describe our physical setup and the components of the setup in a previous publication [5]. Briefly, the air is blown on an evaporative pad in a square duct. The evaporative pad is continuously kept saturated with dripping water using a peristaltic pump. The phase transition in the evaporative pad causes the dry-bulb temperature to drop at the downstream of the pad. While in our previous study we did not have control over fan speed, in the current study, we used a 4-wire fan to allow students to control and acquire fan speed. A 2-wire fan uses power and ground terminals. In a 4-wire fan, the third wire is the tachometer signal, and the fourth wire is for fan speed control using pulse-width modulation. We used a fan controller circuit board (EMC2101, Adafruit Industries, NY) to control the fan. We used two Raspberry Pi 4 computers in the setup. A Raspberry Pi was connected to the fan controller board, while the second one controlled a camera. Using a single Raspberry Pi for both tasks would assert significant computational strain and could cause delays. Therefore, we used two Raspberry Pi boards. An Arduino MKR Wi-Fi

1010 board was utilized to acquire humidity and temperature data from a DHT22 temperature/humidity sensor, which deemed ideal because this sensor could provide both temperature and humidity simultaneously. Both Arduino and Raspberry Pi boards were connected to our university Wi-Fi for fan control, transmitting live-stream and sensor data. Figure 1a shows a schematic of the physical setup. The setup was kept in a closed room with restricted access while students used the setup.

Internet of Things Setup

We used ThingsBoard, an open-source IoT platform. While an instance of ThingsBoard was installed on our institution's servers, it is possible to use ThingsBoard over the cloud for no cost. The no-cost community edition was deemed sufficient for the exercise we describe here. We used Message Queuing Telemetry Transport (MQTT), which is a lightweight publish/subscribe network protocol, for communication between "things" in our setup and users (students). In this protocol, ThingsBoard is an MQTT broker (server) and connected devices in our setup are clients that publish/subscribe to topics in the server. For instance, we create a topic "temperature," and the Wi-Fi-connected Arduino — with DHT temperature sensors attached — publishes temperature data to this topic. The transmitted data packet, the payload, is sent using JavaScript Object Notation which involves attribute-value pairs. In the case of fan speed control, a Wi-Fi connected Raspberry Pi that is connected to the fan controller is subscribed to the topic "…/request" and listens for a set value for the fan speed. Users (students) set the fan speed on the ThingsBoard user interface. ThingsBoard interface is accessible to students by a public URL. We

summarize the data transfer types and direction between “things” and the server in Figure 1b.

The coding we used for Arduino and Raspberry Pi are given in appendices A and B, respectively.

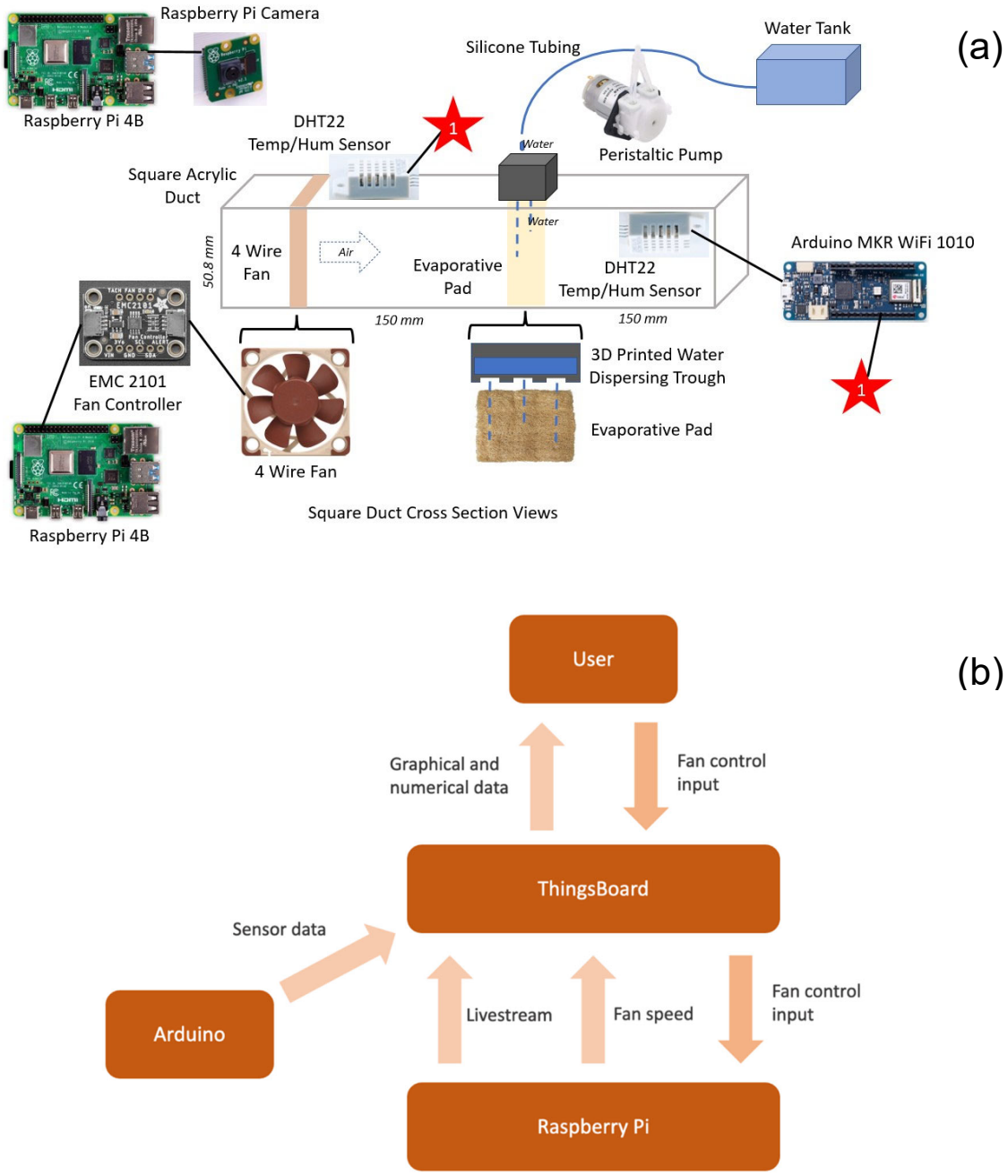

Figure 1. Schematics of the experiment. In (a) we show the physical objects in the evaporative cooler setup. In (b) the flow of information between physical devices, server, and the user is shown.

Livestream of the Setup

While running, the experimental setup was livestreamed to the ThingsBoard dashboard via a Wi-Fi-connected Raspberry Pi with an attached camera (Raspberry Pi Camera Module 2). It allowed students to view the experiment in real-time as they acquired data and controlled the fan. This was accomplished by programming a Raspberry Pi to stream live video through its port 8000. This video feed could then be accessed from a unique URL dependent on the Raspberry Pi's IP address; the URL format was as follows: http://**<Pi_IP_Address>**:8000 [6]. However, this web server was only accessible by devices on the same WiFi network as the Raspberry Pi itself. To allow anyone to view it, this URL was then embedded within the ThingsBoard dashboard by setting the widget's image source as http://**<Pi_IP_Address>**:8000/stream.mjpg. This last part of the URL informs ThingsBoard that the video is a streamed MJPG, or a Motion JPG — a format that has lower memory and processing requirements than other video compression formats. While this URL on its own was local to the network that the Raspberry Pi was connected to, embedding it within the ThingsBoard dashboard essentially allowed students to view a copy of the stream, as if through a window. Therefore, it enabled students to view the livestream on the dashboard without requiring access to the initial video streaming web server. However, this may have had effects on the stream's latency; we are still experimenting with ways to decrease the lag to make the video as real-time as possible for students.

Scheduling System

We provide students a ThingsBoard URL to let them view a dashboard that visualizes/tabulates sensor data and has controls for fan speed. When a user has access to the dashboard, they can control the fan speed. However, multiple users may try to control the fan concurrently if users are not scheduled for use independently. To allocate experiment time to multiple users, we used a free online scheduling system, Calendly. Briefly, we provide students with a link that will allow them to reserve a one-hour time block to use the setup. We used 30 minutes break between consecutive sessions.

Expected Output of the System

In this section, we report the characteristics of our setup relative to fluid dynamics and thermodynamics. While students do not perform the procedure we describe in this section, we believe that the information we provide here is useful for other educators who would like to employ the setup described here. The fan controller we used in the setup allows the user to control the speed of the fan. A user can change the fan speed from 0% to 100%. However, percentile fan speed allows no comparison between different setups. We first measured the volumetric flow rate downstream of the evaporative pad. We utilized a commercially available hot wire anemometer (Wind Sensor Rev. P, Modern Device, RI). The anemometer was placed at the center point of the square duct. Data were acquired using an Arduino Uno. The airspeed measured at 0% to 100% set fan speed with 10% increments. Table I tabulates the air velocity data with different set fan speeds. Air velocity did not fluctuate significantly over time in the

given resolution. Changing fan speed from 0% to 10% and 80% to 100% did not change the air velocity.

Table I. Air velocity at the center point of the duct.

Percentile Rated power	0	10	20	30	40	50	60	70	80	90	100
Centerline Velocity [mph]	0	0	0.05	0.27	0.37	0.58	0.79	0.97	1.09	1.09	1.09

Next, we calculated the volumetric flow rate in the square duct assuming laminar pressure-driven flow for square ducts ($Re \ll 2300$ in our setup). The air velocity at the center of the duct is [7],

$$u = \frac{dP}{dx}\frac{h^2}{8\mu} - \frac{dP}{dx}\frac{4h^2}{\mu\pi^3}\sum_{n=1}^{\infty}\frac{2\sinh(0.5\,\beta_n h)}{\sinh(\beta_n h)}\sin(0.5\,\beta_n h), \quad (1)$$

where,

$$\beta_n = \frac{(2n-1)\pi}{h}. \quad (2)$$

In equation (1), $\frac{dP}{dx}$ is pressure gradient in the duct, h is the inner dimension of the square duct, and μ is dynamic viscosity. Using equation (1) and measured center point velocity, the pressure gradient is calculated. Finally, the volumetric flow rate, Q, is calculated using,

$$Q = \frac{dP}{dx}\frac{h^4}{12\mu} - \frac{dP}{dx}\frac{16h^4}{\mu\pi^5}\sum_{n=1}^{\infty}\frac{1}{(2n-1)^5}\frac{\cosh(\beta_n h)-1}{\sinh(\beta_n h)}. \quad (3)$$

The resulting flow rate is shown in Figure 2.

Next, we calculated the cooling efficiency at different airflow rates using the below formula [5],

$$\varepsilon_e = \frac{T_{d,room}-T_{d,cooler}}{T_{d,room}-T_{w,room}}\ . \quad (4)$$

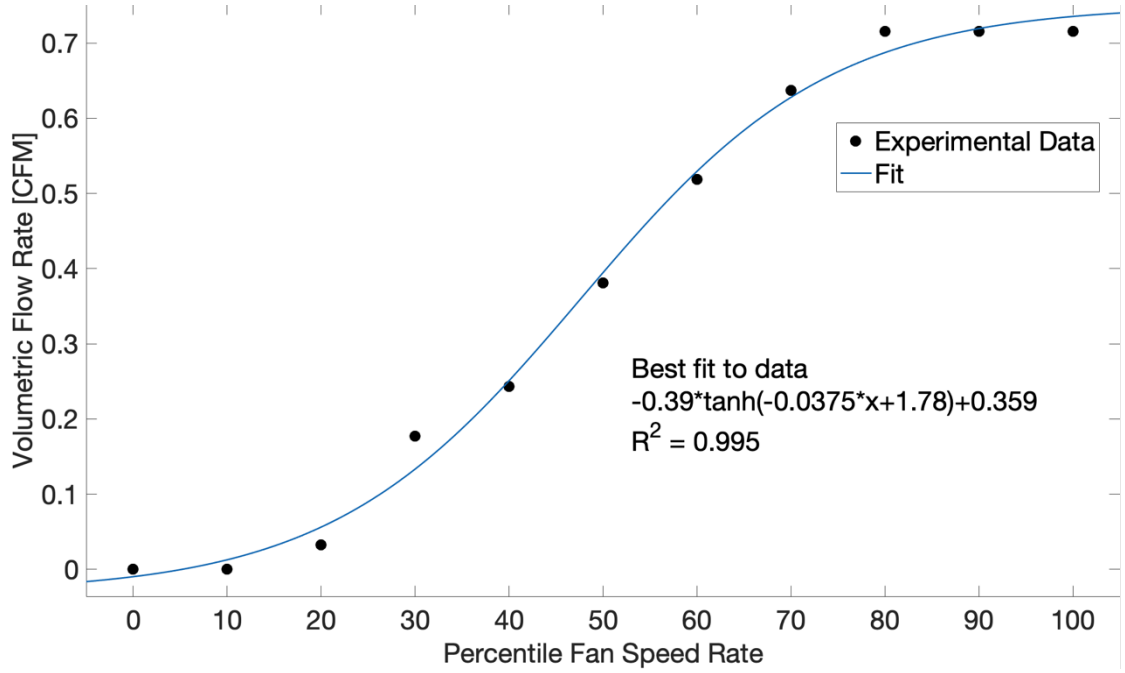

Figure 2. Volumetric flow rate [CFM] in the square duct as a function of different fan speed settings. The fan controller board allows users to set the fan speed at different percentiles. Measurements (circular markers) were conducted between 0% and 100% fan speed with 10% increments.

In Eq. (4), $T_{d,room}$ and $T_{d,cooler}$ are the dry-bulb temperatures for room air and air downstream of the evaporative pad, respectively. $T_{w,room}$ is for the wet-bulb temperature of the room air. Efficiency calculated using equation (4) changes between 0 and 1. We measured the relative humidity and dry-bulb temperature in room air and downstream of the evaporative pad. These data were acquired at set fan speeds between 10% and 100% with 10% increments. Table II tabulates the temperature and humidity data. Cooling efficiency was next calculated at these set fan speeds using equation (4). This equation required the wet-bulb temperature of room air, and this value was obtained using a psychrometric chart. A plot involving the cooling efficiency as a function of air volumetric flow rate is given in Figure 3. We report an increase in efficiency with an increasing flow rate. A theoretical study, such as those in [8], [9], is needed for the given evaporative cooler to understand the relationship between the efficiency and the airflow rate. The

efficiency could decrease at higher flow rates than those given in this study, as water saturation of air will be constant after a critical velocity. We also note that the room that had the experimental setup did not have precise temperature and humidity control, and therefore, the thermodynamic properties of cooled air and efficiency values will change in repetitions and replications of this experiment. However, we expect the same behavior for cooling efficiency vs. volumetric flow rate.

Table II. Dry-bulb temperature and relative humidity downstream from the cooler

Percentile Rated power	20	30	40	50	60	70	80
Temperature [°C]	22.7	22.5	22.4	22.4	22.3	22.3	22.1
Relative Humidity	62.5%	58.1%	56.2%	54%	53.3%	51%	50.6%

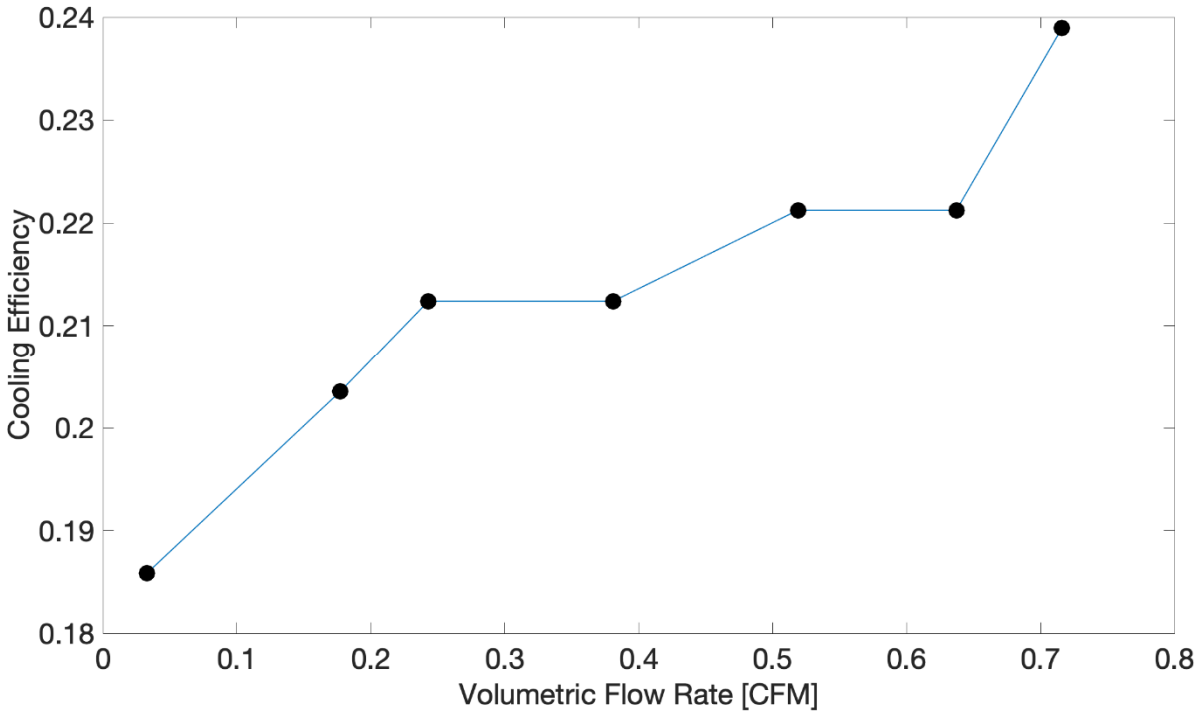

Figure 3. Cooling efficiency (circular markers) as a function of Volumetric flow rate [CFM] in the square duct. Fan speed was varied between 20% and 80% with 10% increments, as volumetric flow rate only changed in between these rated speeds.

Methods to Measure Learning

We offered evaporative cooler exercise as an assignment in an engineering experimentation course in a private technological university. The junior-level course had 90 registered engineering students. We offered this exercise as an optional exercise because scheduling 90 students is a challenge, and our objective in this study was to test the experimental setup and the ThingsBoard interface. By creating the optional exercise, we could form a focus group from which we can get detailed feedback on our setup.

We asked students to calculate cooling efficiency at least for three different fan speeds. Students used equation (4) above and a psychrometric chart to calculate the efficiency. We list our learning objectives below.

- Use the psychrometric chart to determine wet-bulb temperature using other thermodynamic properties.
- Determine cooling efficiency for an evaporative cooler at changing fan speed.

While we did not design this learning activity to involve Internet of Things in our learning objectives, students are exposed to Internet of Things technology in this activity. Students are aware of the capabilities of Internet of Things after completing this exercise.

Students are asked to complete a brief survey at the end of this exercise. The survey questions are as the following.

- Q1: Please rate using the Likert scale. Evaporative cooler activity helped me learn to use a psychrometric chart.

Likert Scale: 0 --> Do not agree 10 --> Strongly agree

- Q2: Please rate using the Likert scale. Evaporative cooler activity helped me learn how evaporative cooling efficiency changes by changing fan speed.

  Likert Scale: 0 --> Do not agree 10 --> Strongly agree
- Q3: This activity was interesting and motivating. (1 – minimum and 10 – maximum)
- Q4: The IoT lab exercise (evaporative cooler) was designed such that you could complete an experiment at home. How likely you would do a similar experiment from home vs. in-class/laboratory:

  Likert Scale:

  0 --> Unpleasant (don't like doing IoT exercise at home, want to see the setup in real-time)

  10 --> Pleasant (like doing IoT exercise at home)
- Q5: If you were to change this experiment, what would you change?

Results

We had five students completing the exercise and giving us feedback on the setup. We report student performance in Table III below. While all students were able to change the fan speed and calculate an efficiency value for each fan speed, three students did not use accurate values for the wet-bulb temperature in the efficiency equation. This resulted in not having a correct conclusion on the relationship between fan speed and cooling efficiency. One student (Student C) used accurate wet-bulb temperature values but reported a decreasing cooling efficiency for the highest fan speed, which is not in concert with the behavior we report above.

Table III. Summary of student performance in evaporative cooler experiment. A "+" sign indicates proficiency and a "–" sign indicates a deficiency in the stated area.

	Wet-Bulb Look-up	Calculate Efficiency	Fan Speed Variation	Conclusion
Student A	-	+	+	-
Student B	+	+	+	+
Student C	+	+	+	-
Student D	-	+	+	-
Student E	-	+	+	-

The responses to Q1- Q4 of the survey are given in Table IV. In the Q5 response, two students reported slow-working and in some instances non-responsive ThingsBoard dashboard. In our test runs, we also experienced a slow dashboard. We haven't tested if this is caused by the Wi-Fi signal in the laboratory, microcontroller bandwidth, or any other reason. One student found the livestream low quality and dark. These reasons might explain the low mean and high standard deviation of the responses to Q4, as students appreciate almost-real-life type controls and high-quality streaming.

Table IV. Summary of student survey responses. Std in the last row stands for standard deviation.

	Q1	Q2	Q3	Q4
Student A	10	10	9	10
Student B	4	7	6	2
Student C	8	10	8	10
Student D	8	8	9	5
Student E	6	6	7	8
Mean	7.2	8.2	7.8	7
Std	2.2	1.7	1.3	3.4

Conclusions

While we could allow students to participate in an experiment actively using Internet of Things technologies, we believe there is room for further improvement. First, students can benefit from several exercise problems where students calculate wet-bulb temperature and use it in the cooling efficiency calculations. Livestreaming needs improvements. A greater pixel count of the video frames and better illumination of the experimental setup are crucial. We used a low-cost Raspberry Pi camera in this study; however, there are higher-end cameras available for Raspberry Pi. Lastly, we will work on the responsiveness of the ThingsBoard dashboard. With the current setup, the dashboard can be unresponsive for several seconds. This could be due to the network connectivity of the microcontrollers/dashboard users or the way the microcontrollers are programmed to publish data to the server. Troubleshooting is needed to find the exact reason.

References

[1] A. Sadik, T. Ortelt, C. Pleul, C. Becker, S. Chatti and A. Tekkaya, "The challenge of specimen handling in remote laboratories for Engineering Education", in *12th International Conference on Remote Engineering and Virtual Instrumentation*, Bangkok, Thailand, 2015, pp. 180-185.

[2] M. Sierra Apel, F. Odebrett, C. Paz and N. Perozo, "Multi-phase flowloop remote laboratory", in *17th International Conference on Remote Engineering and Virtual Instrumentation*, University of Georgia, Athens, Georgia, USA, 2020, pp. 184-192.

[3] R. Franco-Vera, X. Chen and W. Li, "Programmable remote laboratory for mobile robots", in *17th International Conference on Remote Engineering and Virtual Instrumentation (REV)*, University of Georgia, Athens, Georgia, USA, 2020, pp. 75-81.

[4] S. Li, E. Freije and P. Yearling, "Monitoring 3D printer performance using Internet of Things (IoT) application", in *2017 ASEE Annual Conference & Exposition*, Columbus, Ohio, 2017.

[5] A. Sabuncu, J. Sullivan, K. Thornton and M. Mughal, "BYOE: An evaporative cooler with virtual connectivity", in *2021 ASEE Virtual Annual Conference Content Access*, Virtual Conference, 2021.

[6]"Video streaming raspberry pi camera", *RandomNerdTutorials*, 2017. [Online]. Available: https://randomnerdtutorials.com/video-streaming-with-raspberry-pi-camera/. [Accessed: 06- Feb- 2022].

[7] F. White and J. Majdalani, *Viscous fluid flow*, 4th ed. New York: McGraw-Hill, 2021.

[8] J. Wu, X. Huang and H. Zhang, "Theoretical analysis on heat and mass transfer in a direct evaporative cooler", *Applied Thermal Engineering*, vol. 29, no. 5-6, pp. 980-984, 2009. Available: 10.1016/j.applthermaleng.2008.05.016 [Accessed 6 February 2022].

[9] A. Fouda and Z. Melikyan, "A simplified model for analysis of heat and mass transfer in a direct evaporative cooler", *Applied Thermal Engineering*, vol. 31, no. 5, pp. 932-936, 2011. Available: 10.1016/j.applthermaleng.2010.11.016 [Accessed 6 February 2022].

Appendix A: Thingsboard HTML Livestreaming Code

```
<div class='card'>
  <img style="height: 100%"
src="http://130.215.223.226:8000/stream.mjpg">
</div>
```

Appendix B: Raspberry Pi Livestreaming Code

```
# Source code from the official PiCamera package
# http://picamera.readthedocs.io/en/latest/recipes2.html#web-
streaming
# code from https://randomnerdtutorials.com/video-streaming-
with-raspberry-pi-camera/
# Modified by Kerri Thornton at Worcester Polytechnic Institute

import io
import picamera
import logging
import socketserver
from threading import Condition
from http import server

PAGE="""\
<html>
<head>
<title>Raspberry Pi - Surveillance Camera</title>
</head>
<body>
<center><h1>Raspberry Pi Livestream</h1></center>
<center><img src="stream.mjpg" width="640"
height="480"></center>
</body>
</html>
"""

class StreamingOutput(object):
    def __init__(self):
        self.frame = None
        self.buffer = io.BytesIO()
        self.condition = Condition()

    def write(self, buf):
        if buf.startswith(b'\xff\xd8'):
```

```
            # New frame, copy the existing buffer's content and
notify all clients it's available
            self.buffer.truncate()
            with self.condition:
                self.frame = self.buffer.getvalue()
                self.condition.notify_all()
            self.buffer.seek(0)
        return self.buffer.write(buf)

class StreamingHandler(server.BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.send_response(301)
            self.send_header('Location', '/index.html')
            self.end_headers()
        elif self.path == '/index.html':
            content = PAGE.encode('utf-8')
            self.send_response(200)
            self.send_header('Content-Type', 'text/html')
            self.send_header('Content-Length', len(content))
            self.end_headers()
            self.wfile.write(content)
        elif self.path == '/stream.mjpg':
            self.send_response(200)
            self.send_header('Age', 0)
            self.send_header('Cache-Control', 'no-cache,
private')
            self.send_header('Pragma', 'no-cache')
            self.send_header('Content-Type', 'multipart/x-mixed-
replace; boundary=FRAME')
            self.end_headers()
            try:
                while True:
                    with output.condition:
                        output.condition.wait()
                        frame = output.frame
                    self.wfile.write(b'--FRAME\r\n')
                    self.send_header('Content-Type',
'image/jpeg')
                    self.send_header('Content-Length',
len(frame))
                    self.end_headers()
                    self.wfile.write(frame)
                    self.wfile.write(b'\r\n')
            except Exception as e:
                logging.warning(
                    'Removed streaming client %s: %s',
```

```
                    self.client_address, str(e))
        else:
            self.send_error(404)
            self.end_headers()

class StreamingServer(socketserver.ThreadingMixIn,
server.HTTPServer):
    allow_reuse_address = True
    daemon_threads = True

with picamera.PiCamera(resolution='640x480', framerate=24) as
camera:
    output = StreamingOutput()
    #Uncomment the next line to change your Pi's Camera rotation
(in degrees)
    #camera.rotation = 90
    camera.start_recording(output, format='mjpeg')
    try:
        address = ('', 8000)
        server = StreamingServer(address, StreamingHandler)
        server.serve_forever()
    finally:
        camera.stop_recording()
        camera.close()
```

Appendix C: Raspberry Pi Fan Control Code

```
# This Program illustrates the Server Side RPC on ThingsBoard
IoT Platform
# Paste your ThingsBoard IoT Platform IP and Device access token
# Temperature_Controller_Server_Side_RPC.py : This program
illustrates Server side RPC using a Simulated Temperature
Controller
# Original code from
https://shiyaztech.wordpress.com/2018/08/25/remote-procedure-
calls-rpc-on-thingsboard-iot-platform/
# Modifications by Kerri Thornton of Worcester Polytechnic
Institute

import os
import time
import sys
import json
import random
```

```
import paho.mqtt.client as mqtt
from threading import Thread

import board
from adafruit_emc2101 import EMC2101

i2c = board.I2C()
emc = EMC2101(i2c)

# Thingsboard platform credentials
THINGSBOARD_HOST = 'thingsboard.wpi.edu'       #Change IP
Address
ACCESS_TOKEN = ''
sensor_data = {'speed': 25}

# emc.manual_fan_speed = 25

def publishValue(client):
    INTERVAL = 2
    print("Thread  Started")
    next_reading = time.time()
    while True:
        client.publish('', json.dumps(sensor_data),1)
        next_reading += INTERVAL
        sleep_time = next_reading - time.time()
        if sleep_time > 0:
            time.sleep(sleep_time)

def read_manual_speed():
    sensor_data['speed'] = emc.manual_fan_speed #possibly change
to emc.fan_speed. but is RPM scale
    print(emc.manual_fan_speed) #this should be 20
    print(sensor_data['speed']) #this should be 20 as well, to
match above
    manualSpeed = sensor_data['speed']
    return manualSpeed

# Function will set the speed value in device
def setValue (params):
    sensor_data['speed'] = params
    emc.manual_fan_speed = sensor_data['speed']
    #print("Rx setValue is : ",sensor_data)
    print("Speed Set : ",params,"%")
    print("Actual Speed Set: ",emc.manual_fan_speed, "%")

# MQTT on_connect callback function
def on_connect(client, userdata, flags, rc):
```

```
    #print("rc code:", rc)
    client.subscribe('v1/devices/me/rpc/request/+')

# MQTT on_message callback function
def on_message(client, userdata, msg):
    #print('Topic: ' + msg.topic + '\nMessage: ' +
str(msg.payload))
    if msg.topic.startswith('v1/devices/me/rpc/request/'):
        requestId =
msg.topic[len('v1/devices/me/rpc/request/'):len(msg.topic)]
        #print("requestId : ", requestId)
        data = json.loads(msg.payload)
        if data['method'] == 'getValue':
            #print("getvalue request\n")
            #print("sent getValue : ", sensor_data)
            client.publish('v1/devices/me/rpc/response/' +
requestId, json.dumps(sensor_data['speed']), 1)
        if data['method'] == 'setValue':
            #print("setvalue request\n")
            params = data['params']
            setValue(params)

# create a client instance
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.username_pw_set(ACCESS_TOKEN)
client.connect(THINGSBOARD_HOST,1883,60)

t = Thread(target=publishValue, args=(client,))


try:
    client.loop_start()
    t.start()
    while True:
        pass

except KeyboardInterrupt:
    client.disconnect()
```

Appendix D: Arduino ThingsBoard Sensor Data Code

```
//original code from https://www.hackster.io/ahmadradhy/mqtt-
protocol-with-thingsboard-cloud-using-arduino-mkr1010-3a8cdb
//modified by Kerri Thornton at Worcester Polytechnic Institute


#include <Arduino.h>
#include <SPI.h>
#include <WiFiNINA.h>
#include <PubSubClient.h>
#include <ArduinoJson.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>

void setup_wifi();
void reconnect();

struct dataSensor
{
  float temperature = 0.0f;
  float humidity = 0.0f;
};

static char payload[256];
static dataSensor data;
StaticJsonDocument<256> doc;

#define DHTPIN2 2
#define DHTPIN3 3
#define DHTTYPE DHT22

DHT dht1(DHTPIN2,DHTTYPE);
DHT dht2(DHTPIN3,DHTTYPE);

#define TOKEN "ArcPyTest"
#define DEVICEID "131a2910-543a-11eb-a040-3fd57c5d1101"

const char broker[] = "";
const char publishTopic[] = "";
const char* ssid = "";
const char* password = "";

WiFiClient mkr1010Client;
PubSubClient mqtt(mkr1010Client);
```

```
long lastData = 0;

void setup() {
  Serial.begin(9600);
  dht1.begin();
  dht2.begin();
  setup_wifi();
  mqtt.setServer(broker, 1883);
}

void loop() {

  if (!mqtt.connected()) {
    reconnect();
  }

  mqtt.loop();

  float t1 = 0.0F;
  float h1 = 0.0F;
  float t2 = 0.0F;
  float h2 = 0.0F;

  t1 = dht1.readTemperature();
  h1 = dht1.readHumidity();
  t2 = dht2.readTemperature();
  h2 = dht2.readHumidity();

  if (isnan(t1) || isnan(h1)) {
    Serial.println("Failed to read sensor 1");
  }

  if (isnan(t2) || isnan(h2)) {
    Serial.println("Failed to read sensor 2");
  }

  data.temperature = t1;
  data.humidity = h1;
  //need to figure out how to set this for sensor 2

  //doc["temperature"] = data.temperature;
  //doc["humidity"] = data.humidity;

  doc["temperature1"] = t1;
  doc["humidity1"] = h1;
  doc["temperature2"] = t2;
  doc["humidity2"] = h2;
```

```
  serializeJsonPretty(doc, payload);
  long now = millis();

  if (now - lastData > 10000) {
    lastData = now;
    Serial.println(payload);
    mqtt.publish(publishTopic, payload);
  }
}

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);

  while( WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void reconnect(){

  while(!mqtt.connected()) {

    Serial.print("Attempting MQTT connection ....");

    if (mqtt.connect(DEVICEID, TOKEN, NULL)) {
      Serial.println("Connected to MQTT Broker");
    }

    else {
      Serial.print("failed, rc=");
      Serial.print(mqtt.state());
      Serial.println("try again in 5 second");
      delay(5000);
    }
  }
}
```