A Simulation of Fuel Consumption by Hybrid and Diesel TCAT Buses

Marlene Berke (mdb293), Tennyson Bardwell (ttb33), Jane Du (zd53)[1]

[1] *Cornell University*

(Dated: November 2017)

The TCAT is the primary bus service operating in Tompkins county. Our task is to develop a strategy to make use of 8 new hybrid diesel-electric busses so as to reduce the TCAT's fuel consumption. The goal is to compare fuel consumption of diesel-only and hybrid busses on each of 6 routes and determine for which routes using hybrid busses yields the most fuel savings. To that end, we compare the fuel consumption of the two types of busses and optimize for the maximum amount of fuel saved by switching to a hybrid bus on a route, based on our simulation constructed using reasonable parameters and restrictions. We examine the robustness of our suggestion to variation in parameter of the simulation. We conclude that allocating 5 hybrid busses to route 82, 1 to route 81, 1 to route 15, and 1 to route 11 would maximize fuel efficiency.

I. INTRODUCTION

The TCAT's fuel efficiency is desirable for many reasons. First, optimizing fuel use would decrease the cost of operating the TCAT. The TCAT is subsidized by taxes (citation), so reducing the cost of operation could save taxpayer money. Second, saving fuel reduces pollution. As proclaimed by many a T-shirt, "Ithaca is Gorges," and we want to preserve that natural beauty by protecting it from pollution.

Since our goal is to calculate the difference in fuel consumption between diesel-only and hybrid busses on each route, we must consider the conditions under which hybrid busses and diesel-only busses behave differently. In other words, our model must capture the conditions of battery usage in hybrid busses. Because battery usage and charging depends explicitly on terrain (diesel must power the bus on steep hills, and the battery charges when braking on downhills), and terrain differs drastically from route to route, our model must include terrain. That motivates the use of real data about the elevation changes along the TCAT routes. Battery usage also depends on planned bus stops and on driving strategy. Since battery usage depends on second-by-second braking or acceleration, a simulation approach is best suited to this problem. Simulation offers the ability to incorporate the relevant details: real terrain data, an approximation of how humans drive, and flexibility to easily examine how parameter changes affect our conclusions.

Our simulation of diesel and hybrid busses running the routes can be described in several pieces. First, in order to model the busses' energy use, we need information on their acceleration and deceleration. The route's terrain determines a major part of that. For that reason, we use real data on the route's terrain and stops. We call this the "route model." Next, we need to model the motion of the bus, parameterized by the rate of acceleration and deceleration of the bus when driven. This is the "driver model." Last, we must model the diesel engine and the hybrid engine's fuel consumption, called the "engine model." The following sections will detail the simulation piece by piece, model by model, including the assumptions and parameters of each.

Finally, using our simulation, we rank the buses based on fuel saved on one round trip on a route on a weekday morning, as close to 7 am as possible. The fuel consumption of the vehicle depends greatly on terrain, so we take that as the primary point of investigation in our model.

II. MODELS

A. Route Model

1. Outline

To gather data about each route, we used a free service called Strava [1]. Strava is a social networking site for runners and bikers which provides a way for these athletes to plan routes, track their performance, and share their activities with friends. Using the mapping tool, we mapped each TCAT route, and then downloaded a .gpx file listing the latitude and longitude coordinates and elevation about every 30 meters along the route. Using Google Maps, we recorded the latitude and longitude of each bus stop as found on the TCAT bus tracker website. We found the point nearest to the bus stop in our Strava data. We verified that these points were within 30 meters of one another. We designated these points as bus stops. Next, we found the arrival time at each stop from the TCAT schedule for a Monday at 7:00am [2]. This allowed us to create a mapping between location and time, at least for the bus stops. The data files for Route 10 and the code for processing this data can be found in Appendix I.

2. Assumptions

- Location and elevation every 30 meters is sufficiently finely grained
- Strava's elevation data is accurate

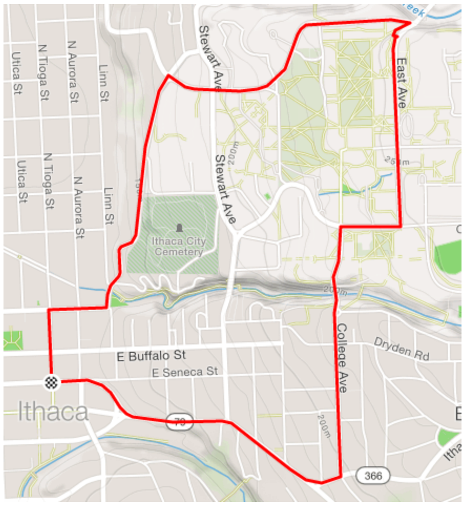

FIG. 1. Route as viewed in Strava

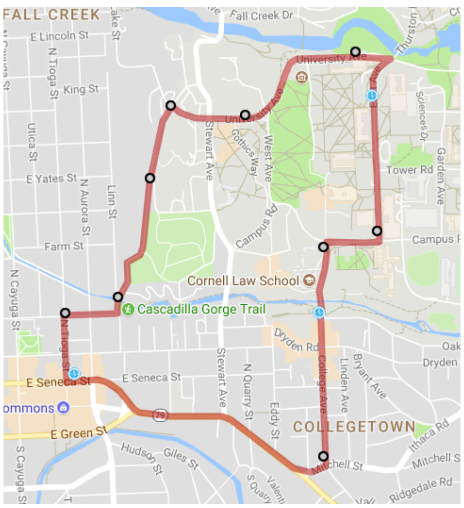

FIG. 2. Route as viewed in TCAT bus tracker

- Google Maps' and Strava's longitude and latitude data is accurate

B. Driver Model

1. *Outline*

We need to describe the motion of the bus. To do so, we made a simple model of human driving. The bus accelerates from stop until it reaches a certain speed. The driver brakes or uses energy as necessary to maintain that "cruise speed" for most of the journey. This same "cruise speed" is maintained both up and down hills, requiring input from the engine at times and braking at other times. The bus decelerates to a stop at the next bus stop. Figure 4 shows a sample map of the target speed vs. time over an interval.

2. *Calculations*

The we assume that the acceleration from a stop and deceleration to a stop is constant. The acceleration constant is a, and deceleration constant is b. These are the only parameters that we arbitrarily fix. The time spent acceleration to cruise speed v is t_1, and the time spent decelerating to rest is t_2. The total time from stop to stop is t. The distance traveled, d, is is simply the area of the trapezoid under the curve in Figure 3. It follows that the distance traveled, d, is described by:

$$d = \frac{vt_1}{2} + \frac{vt_2}{2} + v(t - t_1 - t_2)$$

Furthermore, we know $t_1 = \frac{v}{a}$ and $t_2 = \frac{v}{b}$
By substitution:

$$d = \frac{v}{2}(\frac{v}{a} + \frac{v}{b}) + v(t - \frac{v}{a} - \frac{v}{b})$$

Since d, the distance between two stops, and t, the time the bus has to travel from one stop to another, are known from our Route Model, we solve for v. Between the two roots derived, the higher is usually a degenerate case (in which the speed is too high to be achievable within the given time interval t), so the speed taken is the lower root:

$$v = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Now we have fully described the motion of the bus, treating the engine as a black box.

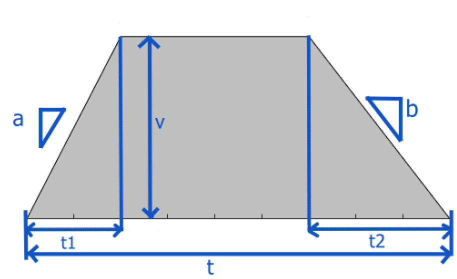

FIG. 3. Velocity vs time over interval between stops

3. *Assumptions*

- Acceleration from a stop and deceleration to a stop are constants. Given the average situation, including sharp braking and sudden flooring of the gas pedal, we assume
- The bus maintains a constant coast speed

- The bus never stops between bus stops (there are no traffic lights or stop signs)
- All TCAT drivers follow the same model of human driving
- Fuel consumed while idling at stops is negligible
- Time spent stopped at a bus stop is negligible
- Diesel and hybrid busses have the same mass, so they accelerate and brake in the same way

4. Parameters

- Acceleration: a =2 m/s^2
- Deceleration: b

C. Engine Model

1. Outline

This model requires some background on how hybrid engines work.

Hybrid busses save fuel by using electricity instead of fuel when possible. A rechargeable battery provides electricity, which can be used to power the bus. However, this electricity does not come for free: it must be generated. While the bus runs on diesel, the battery slowly charges, and while the bus brakes, the battery charges more quickly. There is some limit to how quickly the battery can charge, and there is also a limit to how much charge the battery can store (capacity). Under certain conditions, the bus must rely solely on diesel. Those conditions are when the battery is dead, and when the bus requires power beyond what the battery can provide. The power output of the battery might be exceeded when the bus drives up a very steep hill, or when accelerating from rest. When the power needed is greater than that which the battery can provide, the bus switches from using electricity to using diesel.

To determine the power needed, we must calculate the work, which in turn requires calculating the force that the engine has to exert. Figure 3 shows a free body diagram of the bus driving up a hill. The forces acting on the bus are the force due to the engine, F_{eng}, the force due to gravity, Fg, and the drag force, F_{drag}. When driving down the hill, the diagram and equations are identical. Only the sign of the angle θ changes.

From the free-body diagram in conjunction with Newton's second law of motion, $F = ma$, we have:

$$ma = F_{eng} - F_g \sin(\theta) - F_d$$

The drag follows the following equation, where ρ is the density of the air, v is velocity, C_d is the drag coefficient,

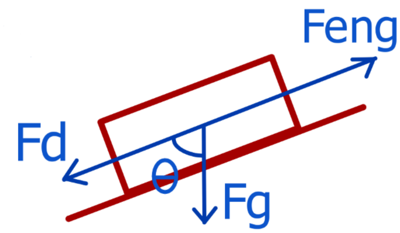

FIG. 4. Free-body diagram of vehicle on slope

and A is the cross-sectional area of the bus orthogonal to velocity:

$$F_d = \frac{1}{2}\rho v^2 C_d A$$

Solving for F_{eng}, we have:

$$F_{eng} = ma + mg\sin(\theta) + \frac{1}{2}\rho v^2 C_d A$$

where m is the mass of the bus, a is the acceleration of the bus, g is the acceleration due to gravity, theta is the angle between level and the slope, and F_{eng} is the force that the engine has to exert on the bus. In our simulation, every time step t is 1 second. Knowing the speed v from our driver model, we know the distance traveled: d = v*t. The work done by the engine Weng is the integral of F_{eng} over d. Since F_{eng} is a constant with respect to d, it reduces to a simple multiplication problem. Our equation for power P_{eng} is as follows:

$$P_{eng} = \frac{dF_{eng}}{t}$$

Again, $t = 1$ for every time step, so we have solved for P_{eng}. When P_{eng} is greater than the maximum power the battery can output, the bus switches from relying on electricity to relying on fuel.

2. Assumptions

- The engine uses electricity whenever possible. Generally, saving energy to be better used at a later point can affect the overall fuel consumption. However, designating the exact times to use diesel despite having battery capacity based on a predetermined route is out of the scope of this problem.
- The engine transitions from running entirely on electricity to running entirely on diesel under two conditions:
  - When the required power output (to maintain speed on a hill or acceleration from rest) exceeds the maximum that the battery can provide

  - When the battery is dead, the fuel spent keeping diesel engine running while stopped or braking is negligible

- The gas tank is large enough that the bus does not have to refuel during a route. This is reasonable given that bus maintenance usually takes place between routes at the main garage
- The engine and battery efficiencies are constant.
- The battery begins at 50 percent of its full charge.

3. Parameters

- Bus mass: $m = 15000$kg
- Maximum capacity of the battery: $c = 4680$kJ or 1.3kilowatt-hours
- Maximum output power of battery: $p = 20000$ Watts
- Maximum charge rate of battery from brakes: $b = 20000$ Watts
- Charge rate of battery from diesel: $d = 1000$ Watts
- Diesel engine efficiency: $de = 180000$ kJ/L
- Electric engine efficiency: $e = 1$ unit
- Density of air: $r = 1.225$ kg/m^3
- Cross-sectional area of bus: $A = 9$ m^2
- Drag coefficient: $C_d = 0.2$

III. ANALYSIS

The below graphs demonstrate the intermediary results used from the model to determine total fuel consumptions on different routes. Route 10 is used as an example. Figure 5 shows the elevation map of the route produced in Strava. Figure 6 shows the battery level over time. It begins at 50 percent and charges reaches full capacity during the latter half of the route, when the bus is going downhill and braking. Figure 7 shows the graph of velocity against time. The points at which the velocity decreases to zero are when the bus stops. The graphs of total fuel consumption over time for both hybrid and diesel buses and for all routes are displayed in Appendix B. The differences are subtle, but the periods of low fuel usage that the diesel-only buses experience are eliminated by the hybrid engine.

See Appendix B for the graphs of fuel consumption over time, and Appendix C for the corresponding simulated battery levels over time for all outes.

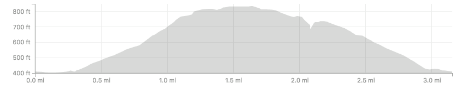

FIG. 5. Elevation map of route 10

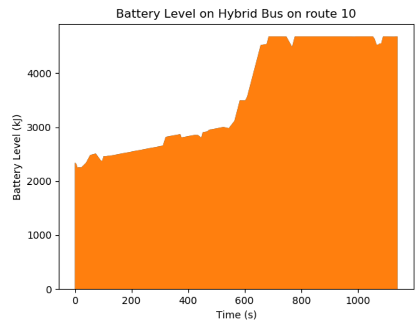

FIG. 6. Battery Level over Time of Hybrid Bus on Route 10

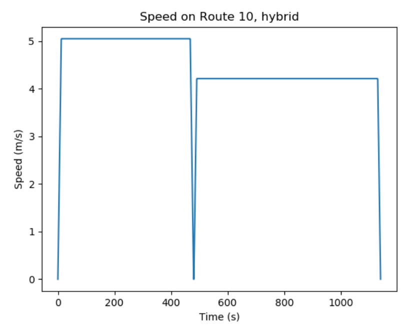

FIG. 7. Speed Profile over Time of bus on Route 10

IV. CONCLUSIONS

The purpose of our simulator was to compare fuel consumption between hybrid and diesel-only buses. Figure I displays the fuel consumption by both bus types on every routes. As the table shows, the hybrid engine saved most fuel (0.36 liters per loop) on route 82. From the TCAT schedule, it appears that 5 buses run that route between 7 and 8am on Monday mornings. Therefore, we advise that 5 of the 8 hybrid buses be assigned to route 82. Only one bus runs routes 81 and 15, the routes for which the hybrid engine results in the next largest savings. Therefore, 1 hybrid bus should be assigned to each. The remaining hybrid bus should be assigned to route 11.

Some parameters in our model, like the coefficient for engine efficiency, for example, affect the savings on all routes equally. Varying these parameters does not effect the results of our model. Others, especially those relating the the battery (like the maximum capacity of the battery), affect the fuel savings, but do not necessarily affect the relative fuel savings across the routes. Because of these observations, we are reasonably confident that small parameter changes will not change the suggested assignments.

TABLE I. Comparison of Fuel Consumption

Route	Fuel Consumed (L) (Diesel Only)	Fuel Consumed(L) (Hybrid)	Fuel Saved (L)
10	1.60	1.56	0.04
11	2.63	2.52	0.09
15	0.40	0.29	0.11
17	0.17	0.09	0.08
81	0.89	0.60	0.29
82	1.91	1.55	0.36

V. STRENGTHS OF CHOSEN MODEL

The strength of our model lies in its ability to accurately compare fuel usage between hybrid and diesel engines on all of the relevant TCAT routes. Accounting for planned bus stops and variation in terrain (uphills and downhills) covers many of the features of a route that would create a difference in fuel consumption between hybrid and diesel engines. Using real, fine-grained data on the changes in elevation lends our model considerable degrees of realism and credibility. Integrating the TCAT bus schedules allows us to model time, which is very important for calculating power, which plays a pivotal role in the engine's "decision" to switch between electricity and diesel.

VI. WEAKNESSES AND FUTURE CONSIDERATIONS

Because of time constraints, we were unable to model several factors. Our model includes starting and stopping at bus stops and variation in power output depending on terrain, but neglects stop signs, speed limits, and stochastic processes like traffic and traffic lights. We gathered real data on the location of the TCAT busses every 5 seconds on the Saturday routes using the TCAT Tracker, but did not have time to incorporate this data into our model. This is a pity, as it would have allowed us validate our "driver model" and incorporate some of the other factors that we had neglected, like traffic. A future model could make use of this incredibly rich data that we unfortunately did not have time to use. Additionally, we compared one loop of a route to one loop of each of the others. These loops differ in length, so in a day, a bus running a longer route might run fewer loops. In that case, comparing the fuel saved in a day on the different routes might be a more meaningful comparison.

Our driver model could be improved without using that data. For example, with more information on how humans drive, we could relax our assumption that drivers try to maintain a constant velocity up and down hills. In fact, we imagine that buses descend hills faster than they ascend them. Additionally, some drivers tend to start and stop more than others. It's conceivable that if the driving style of one TCAT driver is very different than another, one might make better use of a hybrid engine.

A future model could also include engine efficiency. We modeled the efficiency of electrical and diesel engines as constants, and recognize that that is far from the truth. In fact, we believe that the efficiency of a diesel engine depends on the power that it is outputting. There is probably some power for which the diesel engine has maximum efficiency, above and below which the efficiency decreases. For an electric engine, we imagine that the efficiency is probably closer to constant with respect to power output. Therefore, if the efficiencies of both engines were known, the car could strategically switch to the electric engine when the power output is outside of the diesel engine's optimal range. In fact, this is how cars like the Toyota Prius work. Using the battery strategically could increase the difference in fuel consumption even more. If the increase in difference is more pronounced on some routes than others, it might change our suggestions regarding to which routes hybrid buses should be assigned.

The effect of the seasons would also have been interesting to model. Icy hills might require more power to ascend, which would reduce the opportunities for using the electric engine. Using anti-lock brakes on downhills might affect charging the battery from the breaks. Temperature could also affect engine efficiency.

ACKNOWLEDGMENTS

Thanks to all of the professors who made this competition possible, and to Jason Chari, for introducing Marlene Berke to Strava.

[1] N. A. of City Transportation Officials, "Route elevation profile." https://www.strava.com/athletes/26176205.

[2] TCAT, "Tcat stop schedules." https://www.tcatbus.com/ride/schedules-fall-2017/.

[3] N. B. Gov, "School bus acceleration." https://ntl.bts.gov/DOCS/MBTC1054-2.htm.

[4] N. A. of City Transportation Officials, "Vehicle widths and buffers." https://nacto.org/publication/transit-street-design-guide/transit-lanes-transitways/lane-design-controls/vehicle-widths-buffers/.

Appendix A: Letter to The Ithaca Journal

Dear Editor,

Our beloved TCAT provides the primary public transportation in Ithaca. It plays a very important role in our community: bringing hungry college students to Wegmans, the amusement park of Tompkins county, to stock up on Ramen and mac 'n cheese. The TCAT faces an unusual challenge: the very same beautiful terrain that brings us "Ithaca is Gorges" also brings us steep, slippery hills and stomach-plummeting descents. Powering these lumbering buses by diesel cost taxpayer money better spent on education or infrastructure and comes at an environmental cost. For these reasons, we Ithacans want to maximize our fuel efficiency.

To that end, TCAT has added 8 hybrid buses to its fleet. But the question remains: to which routes should the buses be assigned? To solve this problem, our team created a computer simulation of a TCAT buses, hybrid and diesel-only, running every route under consideration. We used real data on the bus stops, bus schedule, and the geography of each route to model the fuel used by hybrid and diesel-only engines. Those factors encompass many of the variables that affect fuel and battery usage. We found that the most fuel would be saved on route 82. We therefore suggest that we use hybrid buses for all of route 82's loops. The remaining hybrid buses should be assigned to routes 81, 15, and 11.

This plan optimally assigns the hybrid buses so as to save diesel. Our plan will both reduce pollution and save money. What's not to like?

Yours truly,
Team TMouse
Marlene Berke, Tennyson Bardwell, and Jane Du

Appendix B: Fuel Consumption

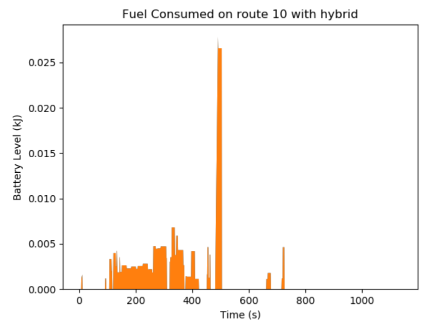

FIG. 8. Fuel of Hybrid Bus over Route 10

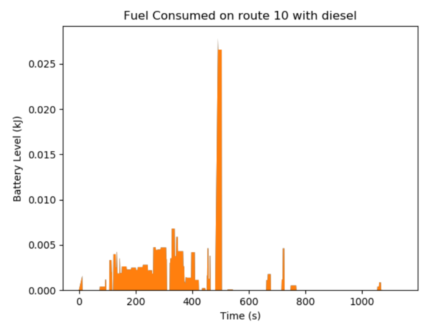

FIG. 9. Fuel of Diesel Bus over Route 10

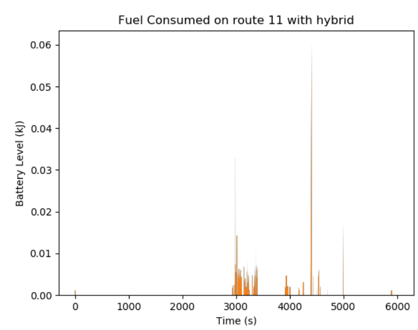

FIG. 10. Fuel of Hybrid Bus over Route 11

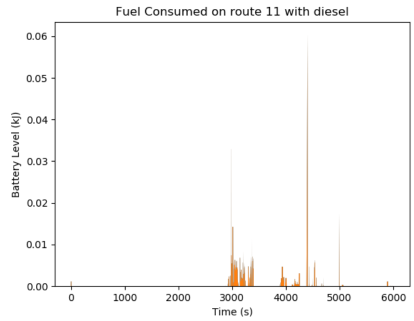

FIG. 11. Fuel of Diesel Bus over Route 11

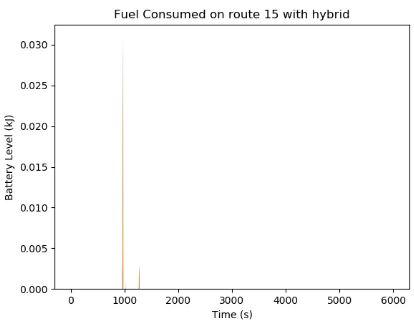

FIG. 12. Fuel of Hybrid Bus over Route 15

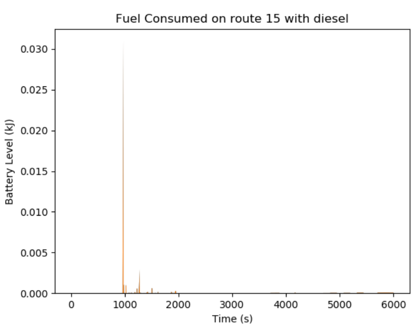

FIG. 13. Fuel of Diesel Bus over Route 10

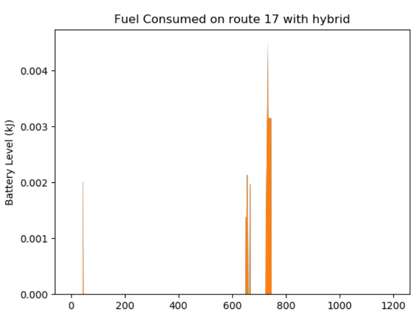

FIG. 14. Fuel of Hybrid Bus over Route 17

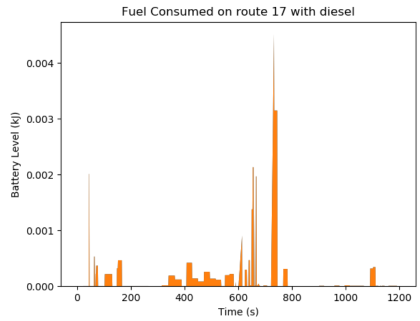

FIG. 15. Fuel of Diesel Bus over Route 17

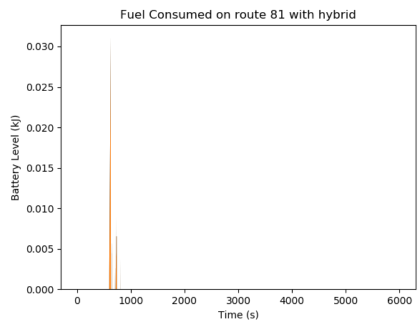

FIG. 16. Fuel of Hybrid Bus over Route 81

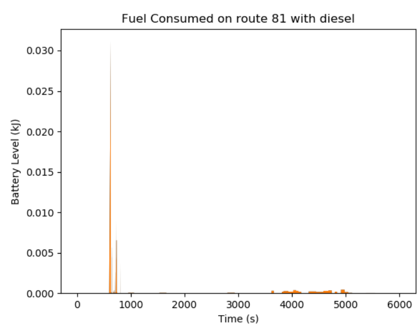

FIG. 17. Fuel of Diesel Bus over Route 81

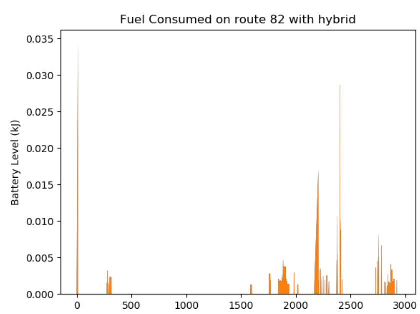

FIG. 18. Fuel of Hybrid Bus over Route 82

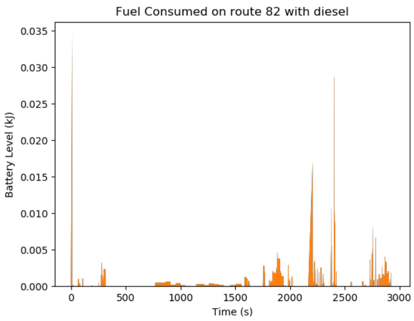

FIG. 19. Fuel of Diesel Bus over Route 82

Appendix C: Battery Level over Time of Hybrid TCAT Bus

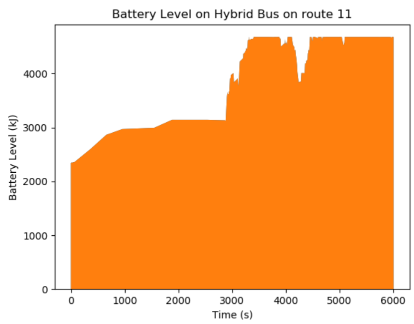

FIG. 20. Battery Level over Time of Hybrid Bus on Route 11

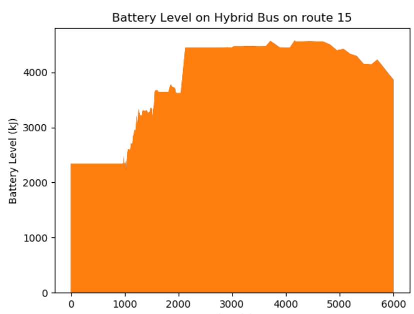

FIG. 21. Battery Level over Time of Hybrid Bus on Route 15

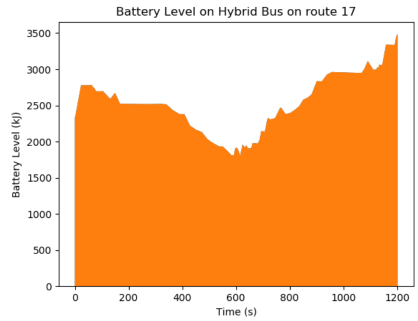

FIG. 22. Battery Level over Time of Hybrid Bus on Route 17

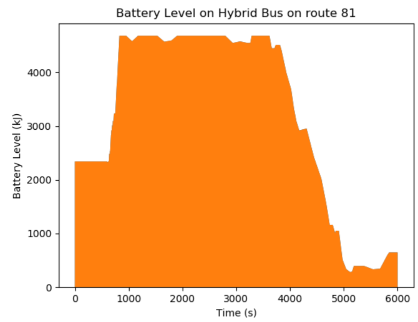

FIG. 23. Battery Level over Time of Hybrid Bus on Route 81

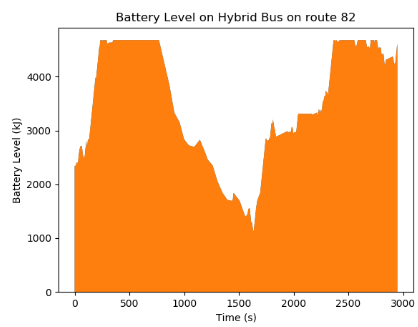

FIG. 24. Battery Level over Time of Hybrid Bus on Route 82

Appendix D: Schema

Below is the structure parsed from the route data taken from Strava and TCAT bus tracker. This is a small sample of the data taken for Route 10.

```
[
    {
        "lat": 42.44046,
        "lon": -76.49686000000001,
        "elevation": 125.0,
        "2d_dist": 0.0,
```

```
        "3d_dist": 0.0,
        "stop": "A"
    },
    {
        "lat": 42.44118666665855,
        "lon": -76.49689666581519,
        "elevation": 124.36000000000001,
        "2d_dist": 80.77582950596556,
        "3d_dist": 80.77836487808364,
        "stop": null
    },
    {
        "lat": 42.441913333305415,
        "lon": -76.49693333248084,
        "elevation": 123.18,
        "2d_dist": 80.7758397806248,
        "3d_dist": 80.78445823464541,
        "stop": null
    }
    ...
]
```


Appendix E: Source Code

Below is the main structure of our model. The full code, with instructions to run, is at https://github.com/janezdu/tmouse

```python
"""
Model hybrid TCAT fuel consumption
"""

import math
import numpy as np
import json
from itertools import chain
from copy import deepcopy
from src.sim import constants as c
import numpy as np


"""
An internal_state object is given in the following form:
    {
        is_diesl: bool
        battery: float
        fuel_used: float
    }

An external_state object is given in the following form:
    {
        grade: float
        speed: float
        acceleration: float
    }
"""

class Path:
    '''a representation of a path through space

    includes elevation, many lat and lon points

    must be a loop unless it is a single path between only two stations

    A must be the first station

    supports tagging points as stations and helper functions on the path'''
    def __init__(self, lst_points):
        '''lst_points is the output of the path importer'''
        self.points = lst_points

    def distance(self, type='3d'):
        '''gets the distance of a point from the start (the first pt)

        type can be 3d or 2d
        '''
        return sum([pt[type+'_dist'] for pt in self.points])

    def _find_points_arount(self, target, type):
        '''returns (a,b,left_over,dist)

        where the pt `target` from the start is between the point a and b,
        and is `left_over` away from a. a and b are `dist` apart
        '''
        bk_dist = 0
        dist = 0
        left_point_index = len(self.points)-2
        for i,pt in enumerate(self.points[1:]):
            bk_dist = dist
            dist += pt[type+'_dist']
            if dist >= target:
                left_point_index = i-1
                break
        l_pt = self.points[left_point_index]
        r_pt = self.points[left_point_index+1]
        return (l_pt, r_pt, target - bk_dist, r_pt[type+'_dist'])

    def grade_at_distance(self, target, type='3d'):
        '''gets the grade at a certain distance
```

```python

        target is the distance from the start of the loop to measure the
        grade at

        type can be 3d or 2d, specifying how the target distance is measured
        '''
        l_pt, r_pt, _, _ = self._find_points_arount(target, type)
        if r_pt['2d_dist']:
            return (r_pt['elevation'] - l_pt['elevation']) / r_pt['2d_dist']
        else:
            return 0

    def location_at_distance(sef, target, type='3d'):
        '''gets the lat and lon at a certain distance

        target is the distance from the start of the loop to measure the
        position at

        type can be 3d or 2d, specifying how the target distance is measured

        returns (lat: float, lon: float, elevation: float)
        '''
        # take a weightage overage of the lat and longitude
        l_pt, r_pt, left_over, size = self._find_points_arount(target, type)
        percent_b = left_over / size
        percent_a = 1 - percent_b

        return (l_pt['lat'] * percent_a + r_pt['lat'] * percent_b,
                l_pt['lon'] * percent_a + r_pt['lon'] * percent_b,
                l_pt['elevation'] * percent_a + r_pt['elevation'] * percent_b)

    def get_intervals(self, stations=None):
        '''returns a smaller path objects for each interval,

        where each interval is a labeled station

        if stations is provided then only those stations are considered
        real stations, (A) must be included

        assumes this is a loop and the first point + station is coppied to the end
        '''
        # index_A = [i for i,x in enumerate(self.points) if x['stop'] == 'A']
        index_A = [i for i,x in enumerate(self.points) if x['stop'] == 'A'][0]
        new_pts = self.points[index_A:] + self.points[:index_A]

        intervals = []
        cur_interval = []
        for pt in new_pts:
            if pt['stop'] \
                    and (stations is None or pt['stop'] in stations) \
                    and cur_interval:

                intervals.append(Path(cur_interval))
                cur_interval = [pt]
            else:
                cur_interval.append(pt)

        cur_interval.append(new_pts[0])
        intervals.append(Path(cur_interval))
        return intervals

    def get_stations(self):
        '''returns a list of marked stations on this path'''
        return [pt['stop'] for pt in self.points if pt['stop']]

    @staticmethod
    def from_file(fp):
        '''reads a json file in the format that the pathing module produces

        fp is a file-like object
        '''
        return Path(json.load(fp))


class Schedule:
    def __init__(self, table):
        '''table should be a list of (label, time) tuples

        time can be a string like '0715', or a relative num of minutes
        '''
        for i,(_,t) in enumerate(table):
            if type(t) == str:
```

```
                table[i][1] = int(t[:2]) * 60 + int(t[2:])
        self.table = table

    def get_stops(self):
        '''gets the stops on this schedule'''
        return [x for x,_ in self.table]

    def duration(self, stop):
        '''returns the time to get to stop from the previous station

        only defined on stations after the first
        '''
        last = None
        for label,time in self.table:
            if label == stop:
                return 60*(time - last)
            else:
                last = time

    def get(self, i):
        '''gets the duration between ()i-1)-th station and i-th station

        0 < i <= num_of_stops
        '''
        assert i != 0
        assert i < len(self.table)
        return self.duration(self.table[i][0])


class SimpleDriver:
    '''A representation of a driver who accelerates, cruses, and
    breaks to arrive on time

    The driver accelerates at a constant speed to reach x, then slows at
    a constant speed to stop at the station exactly on time, varrying x to
    acchieve this.
    '''
    def __init__(self, max_acc=None, max_dec=None):
        self.max_acc = max_acc if max_acc else c.SIMPLE_DRIVER_ACCELERATION
        self.max_dec = max_dec if max_dec else c.SIMPLE_DRIVER_DECELERATION

    def run(self, path, duration):
        '''returns a list of external states, one per second, with duration+1 elements.

        path is a Path object, durration is time in seconds (int)
        stopped in first and last states
        '''
        dist = path.distance()


        # solving a quadratic, ignore larger value
        a = -0.5
        b = duration / ((1/self.max_acc) + (1/self.max_dec))
        c = - dist / ((1/self.max_acc) + (1/self.max_dec))
        max_speed = (-b + math.sqrt(max(0,b**2 - 4 * a * c)))/(2*a)


        # make external state for every time step
        def get_state(t):
            speed = min(self.max_acc * t, max_speed)
            if speed < max_speed:
                # still acc
                acc = self.max_acc
                loc = 0.5 * self.max_acc**2 * t
            else:
                speed = min(speed, self.max_dec * (duration - t))
                if speed < max_speed:
                    # now dec
                    acc = - self.max_dec
                    loc = dist - (0.5 * self.max_dec**2 * (duration - t))
                else:
                    # crusing
                    acc = 0
                    acc_time = max_speed / self.max_acc
                    loc = max_speed * (t - 0.5 * acc_time)

            return {
                    'grade': path.grade_at_distance(loc),
                    'speed': speed,
                    'acceleration': acc
            }
        return [get_state(t) for t in range(duration+1)]


class RoutePlanner:
    '''a class for converting a router's description to a set of states
```

```
    accepts an input of a driver to model how the bus moves
    '''
    def __init__(self, path, schedule, driver):
        '''accepts a path, the accompaning schedule, and a driver function

        the path is a Path object

        the schedule is a schedule object. must be 1 larger than the path object's
        size

        the driver function accepts (sub_path, time) where sub_path is a
        Path object along an interval, and time is the duration the bus has
        to get from start to end
        '''
        assert len(schedule.get_stops()) == len(path.get_stations()) + 1
        self.path = path
        self.schedule = schedule
        self.driver = driver

    def run(self):
        ''''''
        intervals = self.path.get_intervals()
        #print(len(intervals))
        state_intervals = [
                self.driver(sub_path, self.schedule.get(i+1))
                for i,sub_path in enumerate(intervals)
        ]
        return chain(*state_intervals)


class Engine:
    '''a model of how an external_state affects the internal state

    Takes in an external state (accelleration, grade, speed) and a current
    internal state (electricity levels) and computes the next time step,
    one second later
    '''
    def tick_time(self, internal_state, external_state):
        '''this calculates the effect of one time step on the internal_state

        returns a new internal state
        '''
        dt = 1

        new_internal_state = internal_state

        #calculate power needed
        #time is 1 second, d = rt
        a = external_state['acceleration']
        v = external_state['speed']
        dist = v*dt
        grade = external_state['grade']
        theta = np.arctan(grade)
        m = c.MASS

        F = m * c.g * np.sin(theta) + (0.5*c.ro*v**2*c.Cd*c.A) + m * a
        # integrate
        W = F * dist
        # power = work/time. t=1
        power = W/dt

        #if force positive, we're using engine, either battery or diesel
        if F > 0:
            #use battery
            if not internal_state['is_diesl'] and (internal_state['battery'] >= (1/c.ELECTRIC_ENGINE_EFFICIENCY)*W) and (c.POWER_CAP_ELECTRIC >= power):
                new_internal_state['battery'] = new_internal_state['battery'] - (1/c.ELECTRIC_ENGINE_EFFICIENCY)*W
            #use fuel
            else:
                new_internal_state['fuel_used'] = new_internal_state['fuel_used'] + (1/c.DIESEL_ENGINE_EFFICIENCY)*W
                if not internal_state['is_diesl'] and c.BATTERY_CAP > internal_state['battery']:
                    new_internal_state['battery'] += c.BATTERY_CHARGE_FROM_DIESEL
        else:
            #charge battery
            if not internal_state['is_diesl'] and (internal_state['battery'] < c.BATTERY_CAP):
```

```
                new_internal_state['battery']
                ↪ =
                ↪ min(new_internal_state['battery']
                ↪ +
                ↪ c.MAX_BATTERY_CHARGE_RATE*dt,
                ↪ c.BATTERY_CAP,
                ↪ new_internal_state['battery']
                ↪ W)

        return new_internal_state


class Simulator:
    def __init__(self, path, driver, schedule,
        ↪ engine):
        '''creates a new simulation ready to
        ↪ be run for a choice of car

        schedule is a list of times at which
    ↪ A,B,C,etc stations are reached
        '''
        self.path = path
        self.driver = driver
        self.schedule = schedule
        self.engine = engine

    def _run_sim(self, external_states,
        ↪ tick_function, start_state):
        '''a helper function for simulating a
        ↪ list of states

        returns a list of all states from the
    ↪ first to the final
        '''
        internal_state = start_state
        internal_state_list = [start_state]

        for external_state in external_states:

            ↪ internal_state_list.append(deepcopy(internal_state))
            # print(internal_state)
            internal_state =
            ↪ tick_function(internal_state,
            ↪ external_state)

        return internal_state_list

    def run(self, is_diesl,
    ↪ init_electricity=c.BATTERY_CAP/2):
        '''runs the internal state, returns a
        ↪ new internal state
        returns a list of all states from the
    ↪ first to the final'''

        start_state = {
                'is_diesl': is_diesl,
                'fuel_used': 0,
                'battery': init_electricity,
        }
        states = RoutePlanner(self.path,
        ↪ self.schedule, self.driver).run()

        external_states = list(states)

        internal_state_list =
        ↪ self._run_sim(iter(external_states),
        ↪ self.engine.tick_time,
        ↪ start_state)
        # print("===")
        # print(internal_state_list)
        # print("===")
        return {
            "internal_states":
            ↪ internal_state_list[1:],
            "external_states": external_states
        }
        # return self._run_sim(states,
        ↪ self.engine.tick_time,
        ↪ start_state)
```