Lab 2: 1-D Kalman & Bayes Filters

Josephine King, Peter Johnson

***Abstract*—This report details the implementation of two basic state estimation techniques. First, a 1-D Kalman Filter correction step is implemented to track the yaw angle of an IMU. Second, a 1-D Bayes Filter is used to estimate the probability that a car is stopped.**

I. Introduction

FOR Lab 2, we were provided with data from a BNO055 IMU and NuScence's open source vehicle data. The IMU yaw data was obtained by walking in a relatively rectangular path around the Parsons parking lot. The NuScences dataset includes $[x, y, \theta]$ measurements for different vehicles logged at 0.5 second time steps. x and y represent the vehicle's position and θ represents the vehicles' yaw angle.

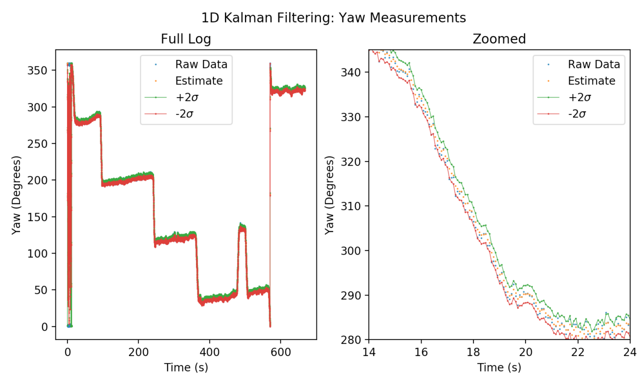

Fig. 1. Yaw measurements showing raw data, estimate, and $\pm 2\sigma$ of the estimate.

II. Kalman Filter

First, we filtered raw yaw data to estimate a yaw angle using a 1D KF correction step which we formulate as follows. We skip the prediction step and simply predict that the current state and variance will not change from the previous time step. The correction step is as follows, where $\overline{x}_t$ and $\overline{\sigma}_t^2$ are the predicted state and variance, K_t is the Kalman gain, z_t and σ_z^2 are the measurement and measurement variance, and $\hat{x}_t$ and $\hat{\sigma}_t^2$ are the corrected state and variance:

$$\overline{x}_t = \hat{x}_{t-1}$$
$$\overline{\sigma}_t^2 = \hat{\sigma}_{t-1}^2$$
$$K_t = \frac{\overline{\sigma}_{t-1}^2}{\overline{\sigma}_{t-1}^2 - \sigma_z^2}$$
$$\hat{x}_t = \overline{x}_t + K_t(z_t - \overline{x}_t)$$
$$\hat{\sigma}_t^2 = \overline{\sigma}_t^2 + K_t \overline{\sigma}_t^2$$

To implement the correction step, we extracted the measurement variance (σ_z^2) using yaw measurements for a stationary yaw angle. We also used σ_z^2 to initialize our first estimate variance σ_0^2. For the results of this filter on one dataset, see Fig 1, which shows raw yaw, estimated yaw, and estimated variance.

III. Bayes Filter

We implemented a Bayes filter that determines the probability that a vehicle i is stopped, given its speed at each time step($p(x_i = stopped|z_i)$). For the prediction step,we used the following conditional probabilities:

$$p(x_{i,t} = stopped|x_{i,t-1} = stopped) = 0.6$$
$$p(x_{i,t} = notstopped|x_{i,t-1} = stopped) = 0.4$$
$$p(x_{i,t} = stopped|x_{i,t-1} = notstopped) = 0.25$$
$$p(x_{i,t} = notstopped|x_{i,t-1} = notstopped) = 0.75$$

We use these conditional probabilities to calculate the predicted belief for each state for time step. For simplicity, we use s for stopped and m for moving (not stopped):

$$\begin{aligned}\overline{bel}(x_{i,t} = s) = p(x_{i,t} = s|x_{i,t-1} = s)bel(x_{i,t-1} = s) + \\ p(x_{i,t} = s|x_{i,t-1} = m)bel(x_{i,t-1} = m) \\ \overline{bel}(x_{i,t} = m) = p(x_{i,t} = m|x_{i,t-1} = s)bel(x_{i,t-1} = s) + \\ p(x_{i,t} = m|x_{i,t-1} = m)bel(x_{t-1} = m)\end{aligned}$$

Next, we incorporate the speed measurement z at that time step in our correction step for each car.

$$bel(x_{i,t} = s) = \eta p(z_{i,t}|x_{i,t} = s)\overline{bel}(x_{i,t=s})$$
$$bel(x_{i,t} = m) = \eta p(z_{i,t}|x_{i,t} = m)\overline{bel}(x_{i,t=m})$$

We incorporated the measurements using conditional probabilities $p(z_{i,t}|x_{i,t})$. We used Car 4 to create the PDF for a stopped car because it was stopped the whole time, and

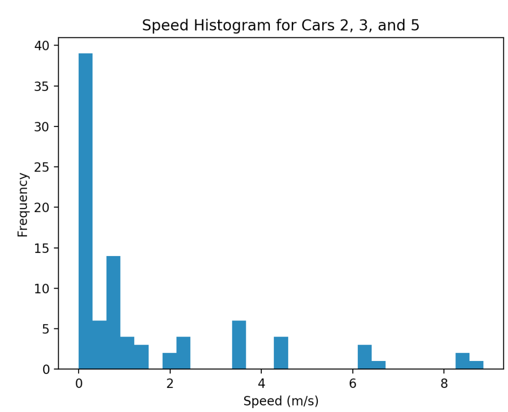

Fig. 2. Histogram of speeds for Cars 2, 3, and 5.

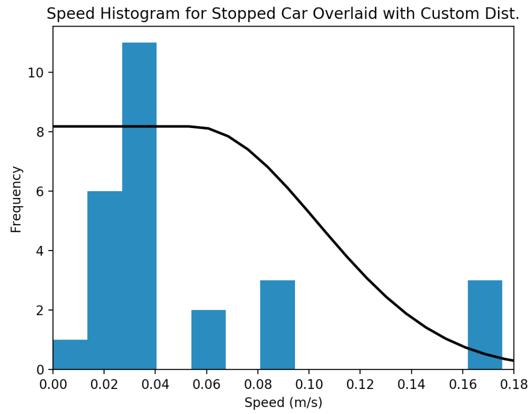

Fig. 3. Speed histogram for Car 4, which was parked the entire time. These data were fit with a custom distribution to make the stopped car model.

we used Car 1 to create the PDF for a moving car because it was moving the whole time. For the stopped car PDF, we combined uniform and half-normal distributions to cover the drop-off in probability as speed increases. For the moving car PDF, we simply fit a normal distribution to Car 1's speed data, which gave a μ of about 8 m/s. This is an approximation that works for our data. It may have been better to use a combination of a uniform and half-normal distribution like we did with the stopped car model. However, the Gaussian fit still works well, because given a measurement of 1 m/s or greater, our models will find that it is much more likely that the car is moving. Our models give us good results, as evidence by Figs 5 and 6.

We see that Car 1 was moving (not stopped) the entire time which matches the video, where it drives ahead of the ego car before exiting the scene. Car 2 is initially stopped as it waits to turn, but then moves until it exits the scene. Car 3 is initially moving towards the ego car, but then stops for Car 2 to turn and then stops again for pedestrians before exiting the scene. Car 4 is parked and stopped the whole time. Car 5 is stopped, moves before it is stopped by pedestrians and then eventually turns.

Based on our filtering, there is a 100% chance that Car 6 is parked for the entire time. There are many parked cars in the video, so it's useful to note that Car 6 is very close to Car 4 and the ego car at the end of the video. Based on this information, we think that Car 6 is the dark grey mini-SUV parked on the right side of the road soon after the red SUV.

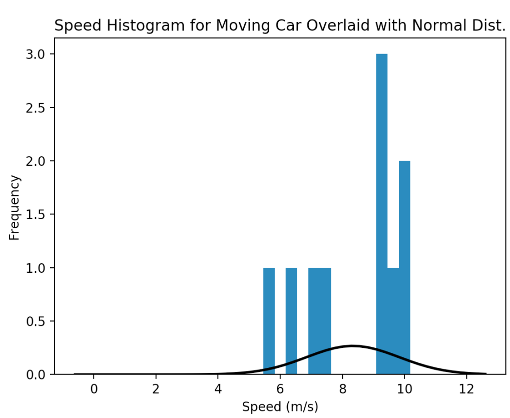

Fig. 4. Speed histogram for Car 1, which was moving the entire time. These data were fit with a normal distribution to make the moving car model.

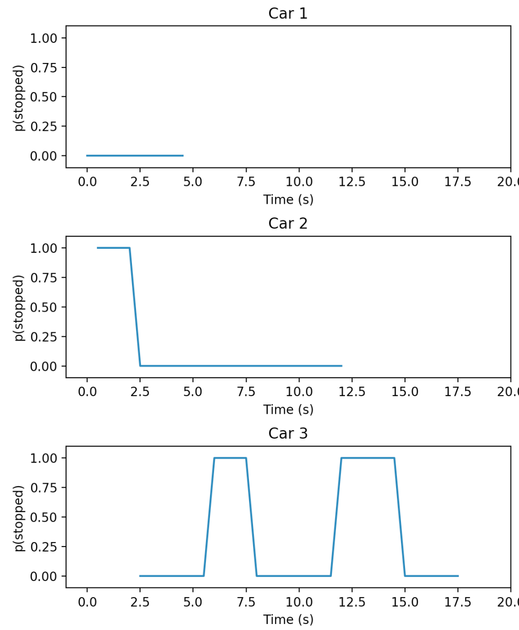

Fig. 5. Probability that Cars 1-3 were stopped, plotted vs. time.

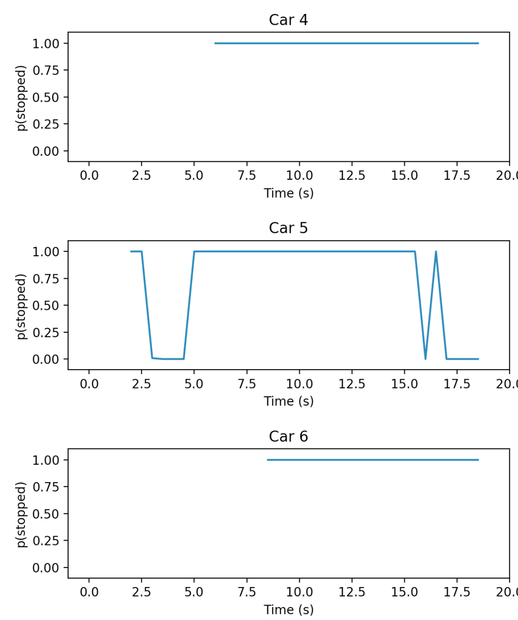

Fig. 6. Probability that Cars 4-6 were stopped, plotted vs. time.

Appendix A
Python Code

A. $run_1D\_KF\_student.m$

```
"""
Peter Johnson and Pinky King based on code by
Author: Andrew Q. Pham
Email: apham@g.hmc.edu
Date of Creation: 2/8/20
Description:
    1D Kalman Filter implementation to filter logged yaw data from a BNO055 IMU
    This code is for teaching purposes for HMC ENGR205 System Simulation Lab 2
"""

import csv
import time
import sys
import matplotlib.pyplot as plt
import numpy as np
import math


def load_data(filename):
    """Load in the yaw data from the csv log

    Parameters:
    filename (str)  -- the name of the csv log

    Returns:
    yaw_data (float list)   -- the logged yaw data
    """
    f = open(filename)

    file_reader = csv.reader(f, delimiter=',')

    # Load data into dictionary with headers as keys
    # Header: Latitude, Longitude, Time Stamp(ms), ...
    # ..., Yaw(degrees), Pitch(degrees), Roll(degrees)
    data = {}
    header = next(file_reader, None)
    for h in header:
        data[h] = []

    for row in file_reader:
        for h, element in zip(header, row):
            data[h].append(float(element))

    f.close()

    yaw_data = data["Yaw(degrees)"]

    return yaw_data


def prediction_step(x_t_prev, sigma_sq_t_prev):
    """Compute the prediction of 1D Kalman Filter

    Parameters:
```

```
    x_t_prev         -- the previous state estimate
    sigma_sq_t_prev -- the previous variance estimate

    Returns:
    x_bar_t          -- the predicted state estimate of time t
    sigma_sq_bar_t  -- the predicted variance estimate of time t
    """

    x_bar_t = x_t_prev
    sigma_sq_bar_t = sigma_sq_t_prev

    return [x_bar_t, sigma_sq_bar_t]


def correction_step(x_bar_t, z_t, sigma_sq_bar_t, sigma_sq_z):
    """Compute the correction of 1D Kalman Filter

    Parameters:
    x_bar_t          -- the predicted state estimate of time t
    z_t              -- the measured state of time t
    sigma_sq_bar_t  -- the predicted variance of time t
    sigma_z          -- the variance of sensor measurement


    Returns:
    x_est_t          -- the filtered state estimate of time t
    sigma_sq_est_t  -- the filtered variance estimate of time t
    """

    Kt = sigma_sq_bar_t/(sigma_sq_bar_t+sigma_sq_z)
    sigma_sq_est_t = sigma_sq_bar_t - Kt*sigma_sq_bar_t
    x_est_t = x_bar_t + Kt*(z_t-x_bar_t)

    return [x_est_t, sigma_sq_est_t]


def wrap_to_360(angle):
    """Wrap angle data to [0, 360]"""
    return (angle + 360) % 360


def plot_yaw(yaw_dict, time_stamps, title=None, xlim=None, ylim=None):
    """Plot yaw data"""
    plt.plot(np.asarray(time_stamps),
             np.array(yaw_dict["measurements"]),
             '.',
             markersize=1)
    plt.plot(np.asarray(time_stamps),
             np.array(yaw_dict["estimates"]),
             '.',
             markersize=1)
    plt.plot(np.asarray(time_stamps),
             np.asarray(yaw_dict["plus_2_stddev"]),
             '.-',
             markersize=1,
             linewidth=0.5)
    plt.plot(np.asarray(time_stamps),
             np.asarray(yaw_dict["minus_2_stddev"]),
```

```python
             '.-',
             markersize=1,
             linewidth=0.5)
    plt.legend(["Raw Data", "Estimate", "+2$\sigma$", "-2$\sigma$"])
    plt.title(title)
    plt.ylabel("Yaw (Degrees)")
    plt.xlabel("Time (s)")
    plt.xlim(xlim)
    plt.ylim(ylim)


def main():
    """Run a 1D Kalman Filter on logged yaw data from a BNO055 IMU."""

    #filepath = ".\"
    filename = "2020-02-08_08_52_01.csv"
    #yaw_data = load_data(filepath + filename)
    yaw_data = load_data(filename)

    """STUDENT CODE START"""
    SENSOR_MODEL_VARIANCE = 1.9273
    """STUDENT CODE END"""

    #  Initialize filter
    yaw_est_t_prev = yaw_data[0]
    var_t_prev = SENSOR_MODEL_VARIANCE
    yaw_dict= {}
    yaw_dict["measurements"] = yaw_data
    yaw_dict["estimates"] = []
    yaw_dict["plus_2_stddev"] = []
    yaw_dict["minus_2_stddev"] = []
    time_stamps = []

    #  Run filter over data
    for t, _ in enumerate(yaw_data):
        yaw_pred_t, var_pred_t = prediction_step(yaw_est_t_prev, var_t_prev)

        # To be explicit for teaching purposes, we are getting
        # the measurement with index 't' to show how we get a
        # new measurement each time step. To be more pythonic we could
        # replace the '_' above with 'yaw_meas'
        yaw_meas = yaw_data[t]
        var_z = SENSOR_MODEL_VARIANCE

        yaw_est_t, var_est_t = correction_step(yaw_pred_t,
                                                yaw_meas,
                                                var_pred_t,
                                                var_z)

        sys.stdout.write("\r Yaw State Estimate: %f    Yaw Measured: %f \n" % (yaw_est_t,yaw_meas))
        sys.stdout.flush()

        sys.stdout.write("Estimated variance: {0}\n\r".format(var_est_t))
        sys.stdout.flush()

        #  Pause the printouts to simulate the real data rate
        dt = 1/13.  # seconds
```

```
        time_stamps.append(dt*t)

        #  For clarity sake/teaching purposes, we explicitly update t->(t-1)
        yaw_est_t_prev = yaw_est_t
        var_est_t_prev = var_est_t

        # Pack data away into yaw dictionary for plotting purpose
        plus_2_stddev = wrap_to_360(yaw_est_t + 2*math.sqrt(var_est_t))
        minus_2_stddev = wrap_to_360(yaw_est_t - 2*math.sqrt(var_est_t))

        yaw_dict["estimates"].append(yaw_est_t)
        yaw_dict["plus_2_stddev"].append(plus_2_stddev)
        yaw_dict["minus_2_stddev"].append(minus_2_stddev)

    print("\n\nDone filtering...plotting...")

    # Plot raw data and estimate
    plt.figure(1)
    plt.suptitle("1D Kalman Filtering: Yaw Measurements")
    plt.subplot(1, 2, 1)
    plot_yaw(yaw_dict, time_stamps, title="Full Log")
    plt.subplot(1, 2, 2)
    plot_yaw(yaw_dict,
             time_stamps,
             title="Zoomed",
             xlim=[14, 24],
             ylim=[280, 345])
    plt.show()

    print("Exiting...")

    return 0


if __name__ == "__main__":
    main()
```

B. 1d_BF_student.m

```
"""
Peter Johnson and Pinky King based on code by
Email: apham@g.hmc.edu
Date of Creation: 2/8/20
Description:
    1D Bayes Filter implementation to filter logged x,y,yaw data from a nuscene
    This code is for teaching purposes for HMC ENGR205 System Simulation Lab 2
"""

import csv
import time
import sys
import matplotlib.pyplot as plt
import numpy as np
import math
from scipy.stats import norm, halfnorm, uniform
import scipy.stats

#Global Variables
#Probability density parameters
```

```
STOP_MU = 0
STOP_STD = 0
MOVE_MU = 0
MOVE_STD = 0

#Conditional Probabilities
#p(X_t = Stopped | x_t_p = Stopped)
pS_S = 0.6
pS_M = 0.25
pM_S = 0.4
pM_M = 0.75

#Offsets for first time index for each car
#Starts with car 1, goes up to 6
time_offsets = [0, 1, 5, 12, 4, 17]

#Time step for nuscene data
dt = 0.5

def load_data(filename):
    """Load in the data from the csv log

    Parameters:
    filename (str)  -- the name of the csv log

    Returns:
    data (float list)   -- the logged car data
    """
    f = open(filename)

    file_reader = csv.reader(f, delimiter=',')

    # Load data into dictionary with headers as keys
    # Header: Latitude, Longitude, Time Stamp(ms), ...
    # ..., Yaw(degrees), Pitch(degrees), Roll(degrees)
    data = {}
    header = next(file_reader, None)
    for h in header:
        data[h] = []

    for row in file_reader:
        for h, element in zip(header, row):
            if element in (None,""):
                continue
            else:
                data[h].append(float(element))
    f.close()

    # Fixing a glitch in importing the time header
    for key in data:
        if "Time" in key:
            data["Time"] = data.pop(key)
            break

    # Add Speed data to the Dictionary
    for j in range(1,7):    #Loop through i=1 to i=6
        x = []
        y = []
```

```
        xname = "X_" + str(j)
        yname = "Y_" + str(j)
        sname = "S_" + str(j)
        data[sname] = []
        x = data[xname]
        y = data[yname]
        for i in range(len(x)-2):
            data[sname].append(math.sqrt((x[i+1]-x[i])**2 + (y[i+1]-y[i])**2)/dt)

    return data

def hist_plotter(data, car_num, dt):
    """Takes in speed data and list of car numbers, and makes a histogram
        of all their speeds
    """
    speed = []

    for i in range(len(car_num)):
        sname = "S_" + str(car_num[i])
        speed_dat = data[sname]
        for j in range(len(data[sname])):
            speed.append(speed_dat[j])

    numbins = int(len(speed)/3)
    plt.hist(speed, bins=numbins)
    plt.xlabel("Speed (m/s)")
    plt.ylabel("Frequency")


def sensor_model_stopped(data, car_num, dt):
    """ Uses car data to create a histogram of vehicle
        speed and then creates a pdf for a stopped car
    """
    speed = []

    for i in range(len(car_num)):
        sname = "S_" + str(car_num[i])
        speed_dat = data[sname]
        for j in range(len(data[sname])):
            speed.append(speed_dat[j])

    # Plot histogram
    numbins = int(len(speed)/2)
    plt.hist(speed, bins=numbins)

    # Fit speeds with a normal distribution
    mu, std = norm.fit(speed)

    # Make piecewise probability distribution function
    # Uniform distribution between 0 and mu calculated for normal
    # dist. After that, just a half norm
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, len(speed))
    p = []
    for i in range(len(x)):
        if (x[i] < mu):
            p.append(norm(mu, std).pdf(mu))
        else:
```

```
            p.append(norm(mu, std).pdf(x[i]))

    plt.plot(x, p, 'k', linewidth=2)
    plt.xlim(0,.18)
    plt.title("Speed Histogram for Stopped Car Overlaid with Custom Dist.")
    plt.xlabel("Speed (m/s)")
    plt.ylabel("Frequency")

    return [mu,std]

def sensor_model_moving(data, car_num, dt):
        """ Uses car data to create a histogram of vehicle
            speed and then create a pdf
            Calculate speed using distance from euclidean change in position
            returns the average and standard devation for gaussian fit of data
        """
        speed = []

        for i in range(len(car_num)):
            sname = "S_" + str(car_num[i])
            speed_dat = data[sname]
            for j in range(len(data[sname])):
                speed.append(speed_dat[j])

        # Plot histogram
        numbins = int(len(speed)/0.3)
        plt.hist(speed, bins=numbins, range=(0,12))

        # Fit speeds with a normal distribution
        mu, std = norm.fit(speed)

        xmin, xmax = plt.xlim()
        x = np.linspace(xmin, xmax, 5*len(speed))
        p = norm.pdf(x, mu, std)

        plt.plot(x, p, 'k', linewidth=2)
        plt.title("Speed Histogram for Moving Car Overlaid with Normal Dist.")
        plt.xlabel("Speed (m/s)")
        plt.ylabel("Frequency")
        return [mu,std]

def p_moving_s(s):
    """ Takes in car speed, returns p(s|moving), which is the probability
        that speed measurement is s if the car is moving
    """
    pdf_val = norm(MOVE_MU, MOVE_STD).pdf(s)
    # cdf integrates over pdf. Put in a high value of 10 to get whole range,
    # then subtract the region less than 0 because those speeds are impossible
    cdf_val = halfnorm(MOVE_MU, MOVE_STD).cdf(10)-norm(MOVE_MU, MOVE_STD).cdf(0) # normalize with cdf
    prob = pdf_val/cdf_val
    return prob

def p_stopped_s(s):
    """ Takes in car speed, returns p(s|stopped), which is the probability
        that speed measurement is s if the car is stopped
    """
    # Make piecewise probability distribution function
```

```python
    # Uniform distribution between 0 and mu calculated for normal
    # dist. After that, just a half norm
    if (s < STOP_MU):
        pdf_val = norm(STOP_MU, STOP_STD).pdf(STOP_MU)
    else:
        pdf_val = norm(STOP_MU, STOP_STD).pdf(s)

    # cdf integrates over pdf. Put in a high value of 10 to get whole range,
    # then subtract the region less than mu and add in the uniform region
    cdf_val = norm(STOP_MU, STOP_STD).cdf(10) + STOP_MU*norm(STOP_MU, STOP_STD).pdf(STOP_MU) - norm(STOP_MU, STOP_STD).cdf(STOP_MU)
    prob = pdf_val/cdf_val # normalize with cdf
    return prob

def bayes_filter_step(b_x_tp_S, b_x_tp_M, s):
    """ Returns the belief (probability) bel_(x_t) for the moving and stopped
        states
        inputs: the previous belief in stopped and moving state, current speed
        output the predicted beliefs
    """
    #Prediction Step
    #bel_bar(x=S) = p(S|S)*p(S) + p(S|M)*p(M)
    bb_x_t_S = pS_S*b_x_tp_S + pS_M*b_x_tp_M
    bb_x_t_M = pM_S*b_x_tp_S + pM_M*b_x_tp_M

    #Correction step
    b_x_t_S = p_stopped_s(s)*bb_x_t_S
    b_x_t_M = p_moving_s(s)*bb_x_t_M

    #Normalize
    norm = b_x_t_S + b_x_t_M
    b_x_t_S = b_x_t_S/norm
    b_x_t_M = b_x_t_M/norm

    return [b_x_t_S, b_x_t_M]

def plot_bayes(data, time_offset, times):
    """ Plots the Bayes filter prediction for a given car's data
        vs. time
    """

    #Initialize beliefs for each state
    bf = []
    b_x_tp_S = 0.5
    b_x_tp_M = 0.5
    for i in range(len(data)):
        # Repeatedly calls Bayes filter step, then plots vs. time
        [b_x_tp_S, b_x_tp_M] = bayes_filter_step(b_x_tp_S, b_x_tp_M, data[i])
        bf.append(b_x_tp_S)

    plt.plot(times[time_offset:time_offset+len(data)], bf)

def main():
    """Run a 1D Bayes filter on logged movement """

    filename = "E205_Lab2_NuScenesData.csv"
    data = load_data(filename)

```

```
    # global variables
    global STOP_MU
    global STOP_STD
    global MOVE_MU
    global MOVE_STD

    # Use car 4 data to develop conditional stopped probabilities
    # p(s_i|x_i = stopped)
    plt.figure(1)
    [STOP_MU,STOP_STD] = sensor_model_stopped(data, [4], dt)
    plt.show()

    # Use car 1 to develop our model for a moving car
    # p(s_i|x_i = moving)
    plt.figure(2)
    [MOVE_MU,MOVE_STD] = sensor_model_moving(data, [1], dt)
    plt.show()

    # Make histogram for cars 2, 3, 5
    plt.figure(4)
    hist_plotter(data, [2,3,5], dt)
    plt.title("Speed Histogram for Cars 2, 3, and 5")
    plt.show()

    # Plot stopped probability for each car vs. time
    plt.figure(5)
    times = data["Time"]

    for i in range(1,4):
        sname = "S_" + str(i)
        speeds = data[sname]
        plt.subplot(3, 1, i)
        plt.ylim(-.1,1.1)
        plt.xlim(-1,20)
        plt.title("Car " + str(i))
        plt.xlabel("Time (s)")
        plt.ylabel("p(stopped)")
        plot_bayes(speeds, time_offsets[i-1], times)
    plt.show()

    plt.figure(6)
    for i in range(4,7):
        sname = "S_" + str(i)
        speeds = data[sname]
        plt.subplot(3, 1, i-3)
        plt.ylim(-.1,1.1)
        plt.xlim(-1,20)
        plt.title("Car " + str(i))
        plt.xlabel("Time (s)")
        plt.ylabel("p(stopped)")
        plot_bayes(speeds, time_offsets[i-1], times)
    plt.show()

    print("Exiting...")

    return 0
```

```python
if __name__ == "__main__":
    main()
```