Kalman Filtering GPS Data

Alec Henderson
MA-326, North Carolina State University

Introduction

Kalman filters are widely used in several fields for predicting the underlying state of a system in the presence of uncertainty and noise. One of the most prominent applications of a Kalman filter is for "dead reckoning," which is the process of extrapolating the current position of a GPS device using sensor data and previous GPS updates. The Kalman filter is uniquely qualified for this task as it can both predict current state and filter out noisy data.

In this paper, we will first explore the Kalman filter from a high level, then dive into the finer details of how the algorithm works. We will then see how we can implement a Kalman filter for making GPS data more accurate using an Arduino and GPS module.

Table of Symbols

Symbol	Name	Dimensions
x	State Vector	n_x
z	Measurement Vector	n_z
F	State Transition Matrix	$n_x \times n_x$
u	Input Vector	n_u
G	Control Matrix	$n_x \times n_u$
P	Estimate Covariance Matrix	$n_x \times n_x$
Q	Process Noise Covariance	$n_x \times n_x$
R	Measurement Covariance Matrix	$n_z \times n_z$
w	Process Noise Vector	n_x
v	Measurement Noise Vector	n_z
H	Observation Matrix	$n_z \times n_x$
K	Kalman Gain	$n_x \times n_z$
n	Discrete Time Index	

Note on Notation

a variable u_n is interpreted as "u at time-step n," and a variable $\hat{x}_{a,b}$ is interpreted as "The estimation of x at time-step a taken at time-step b." For example, $x_{n+1,n}$ is interpreted as a estimation of x's value at time-step $n+1$ taken at time-step n.

Kalman Filter

High Level Overview

The Kalman filter is a recursive algorithm that consists of two main steps: the **prediction step** and the **update step**. The prediction stage utilizes a model of the system and previous state and uncertainty information to produce a new estimate of the current state variable (x) and uncertainty (P). The update stage takes a measurement (z), and updates the state variable using a weighted average (K) which favors estimations with lower uncertainty.

The algorithm is initialized with a guess of initial state and uncertainty $(\hat{x}_0, P_0)$. In principle, this guess could be initialized to anything as the algorithm will eventually converge to an acceptable value (assuming the system remains static). In our GPS implementation, we will take an initial $\hat{x}_0$ as the first measurement from our GPS, and values roughly equivalent to our measurement's error for P [3].

Prediction Step

Two equations make up the prediction step: state extrapolation and estimate covariance extrapolation. The state extrapolation is given by:

$$\hat{x}_{n+1,n} = F\hat{x}_{n,n} + Gu_n$$

where F takes the state of the system at time n and maps it to the state at time $n+1$, which typically involves the elapsed time Δt, and G maps control input at time n to the state at time $n+1$ [1].

The estimate covariance extrapolation is given by:

$$P_{n+1,n} = FP_{n,n}F^T + Q$$

where P and Q both represent error, which can be separated because the error represented in P_n is the error up to time-step n (hence transformation via F), whereas Q is noise accumulated between time-steps n and $n+1$ [3].

Update Step

Three equations make up the update step, the result of which leaves us with the current estimate of our system state.

The first equation computes the "Kalman Gain" (K_n), which can be thought of intuitively as a weighting factor, whose value (or values) is based on how much we "trust" our new measurement (z_n) compared to our previous state estimate ($\hat{x}_{n-1}$) [4]. The equation for Kalman gain is given by:

$$K_n = P_{n,n-1}H^T(HP_{n,n-1}H^T + R_n)^{-1}$$

where H projects our previous covariance (P_{n-1}) into the measurement domain and R adds our measurement uncertainty. Intuitively, we can see that if R is large then the resulting value of K_n is small, resulting in lower "trust" in our measurements.

The second equation uses our new K_n to update our estimate of the current state. This estimation update equation is given by:

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - H\hat{x}_{n,n-1})$$

where the term $(z_n - H\hat{x}_{n,n-1})$ is known as the "innovation" or "measurement residual" [3].

The third and final equation of the update step updates the estimate uncertainty (P_n). This is given by:

$$P_{n,n} = (I - K_nH)P_{n,n-1}(I - K_nH)^T + K_nR_nK_n^T$$

which is sometimes simplified to

$$P_{n,n} = (I - K_nH)P_{n,n-1}$$

which is simpler, but is considered to be numerically unstable [5].

Algorithm

Now that we have described all of the equations used to run a Kalman filter, it is useful to describe how we plan its implementation in the form of a pseudocode algorithm:

1. Initialize $\hat{x}_0, P_0$ (given)
2. set $n = 0$
3. loop:
    1. Prediction step:
        1. compute $\hat{x}_{n+1,n}$
        2. compute $P_{n+1,n}$
    2. $broadcast(\hat{x}_{n+1})$
    3. wait Δt units of time
    4. update $n = n + 1 \implies \hat{x}_{n,n-1} = \hat{x}_{n+1,n}, P_{n,n-1} = P_{n+1,n}$
    5. Update step:

    1. take measurement z_n
    2. compute K_n
    3. compute $\hat{x}_{n,n}$
    4. compute $P_{n,n}$
6. $broadcast(\hat{x}_{n,n})$

The function $broadcast(x, y, z, \dots)$ is used to communicate the values of $x, y, z \dots$ to the rest of the system. Similar behavior could be accomplished in code with a global variable or message system. We use two broadcasts as both the current and future values of $\hat{x}$ could be relevant to the user based on the application. The waiting for Δt is to simulate processing time or waiting on a measurement.

Implementation

My implementation of the Kalman filter in C++ (called the *reckoner*, see Appendix A) is in the form of a templated library that we can instantiate based on n_x, n_z, n_u which are the dimensions of our state vector, measurement vector, and control (or input) vector respectively. The implementation is exactly as shown above with a few exceptions:

- We assume the process noise and measurement covariance are both constant
- Δt is a measured value rather than waited for

Initial Values

Because this implementation is kept as generic as possible, we need to provide the reckoner with $\hat{x}_0, P_0, Q, R, F, G, H$ as $\hat{x}_0, P_0, F, G$ vary based on our dynamical system and Q, R are based on our hardware and process imprecision.

Dynamical System State-Space Derivation

In our implementation, we model a 2D dynamical system where the state comprises position and velocity in the North and East directions. The system evolves over time according to the current velocity, and the control input drives changes in velocity. The state vector x is of the following form:

$$x = \begin{bmatrix} p_N \\ p_E \\ v_N \\ v_E \end{bmatrix}$$

where p denotes position (in meters) and v denoted velocity in meters per second.
Note: The goal of this implementation is to explore Kalman filters, not geodesy. therefore, we

use euclidean distance instead of geodesic length for simplicity. This accuracy of this filter **will** degrade over great distances.

In order to drive state updates, we will use measurements from a compass and accelerometer to get our control input u, which is of the form:

$$u = \begin{bmatrix} a_N \\ a_E \end{bmatrix}$$

where a_N, a_E are acceleration in the North and East direction respectively.

To satisfy the extrapolation equation $\hat{x}_{n+1,n} = F\hat{x}_{n,n} + Gu_n$, we will also need F and G as functions of time. For this, we recall that velocity and acceleration are the respective first and second differentials of position with respect to time, and we get the following of equations:

$$p_N = p_{N_0} + v_N \Delta t + \frac{1}{2} a_N \Delta t^2$$

$$p_E = p_{E_0} + v_E \Delta t + \frac{1}{2} a_E \Delta t^2$$

$$v_N = a_N \Delta t$$

$$v_E = a_E \Delta t$$

which, in our desired state-space form is:

$$\hat{x}_{n+1,n} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_N \\ p_E \\ v_N \\ v_E \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} a_N \\ a_E \end{bmatrix}$$

therefore,

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix}$$

Measurement Covariance

For our implementation we are using the Adafruit Ultimate GPS module which is accurate to three meters according to the manufacturer. Therefore, assuming a standard deviation of $\sigma = 2.694 \times 10^{-5}$ (roughly three meters in degrees), we get a measurement covariance matrix of

$$R = \begin{bmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{bmatrix} = \begin{bmatrix} 7.262 \times 10^{-10} & 0 \\ 0 & 7.262 \times 10^{-10} \end{bmatrix}$$

Process Noise Covariance

We will let the values of Q be a tunable parameter of our reckoner, as we can assume a Gaussian distribution of noise, however we do not know *a priori* the distribution of this noise.

Initial State

In order to get our initial state and covariance ($\hat{x}_0, P_0$), we will take take the first GPS measurement and take it's value as our initial guess, with the goal of our reckoner eventually converging to something more accurate. P_0 will be a tunable parameter, because we do not know *a priori* what the actual state covariance will be. To start, we will assume that

$$P_0 = \begin{bmatrix} 7.262 \times 10^{-10} & 0 & 0 & 0 \\ 0 & 7.262 \times 10^{-10} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

based on our measurement covariance and an assumed error in our velocity of 1 meter per second.

Observation Matrix

H is used to convert our state into to our measurement space so that it can interact with our measurements correctly. In order to do this, we must use H to convert x from meters to degrees.

$$H = \begin{bmatrix} 1/q_{lat} & 0 & 0 & 0 \\ 0 & 1/q_{lon} & 0 & 0 \end{bmatrix}, \quad q_{lat} \approx 111320,\ q_{lon} = 111320 * cos(\hat{x}_{0_1})$$

The value for q_{lon} is determined at runtime based on the cosine of our latitude, as this affects how long one degree of longitude is (longer at the equator than at the poles).

Testing

Stationary Testing

To test my reckoner from a stationary position, I first found the coordinates of my exact position in Google Maps and then ran the Kalman Filter for some time in order to see how the estimated position changed over time. The keen reader will note that the prediction step will have no effect, as no motion in the system means the extrapolated position will be identical to the current one, so we are essentially only testing the update stage here.

Test 1

My first test from my apartment was flawed but showed promise. Plotting some outputted points resulted in the following:

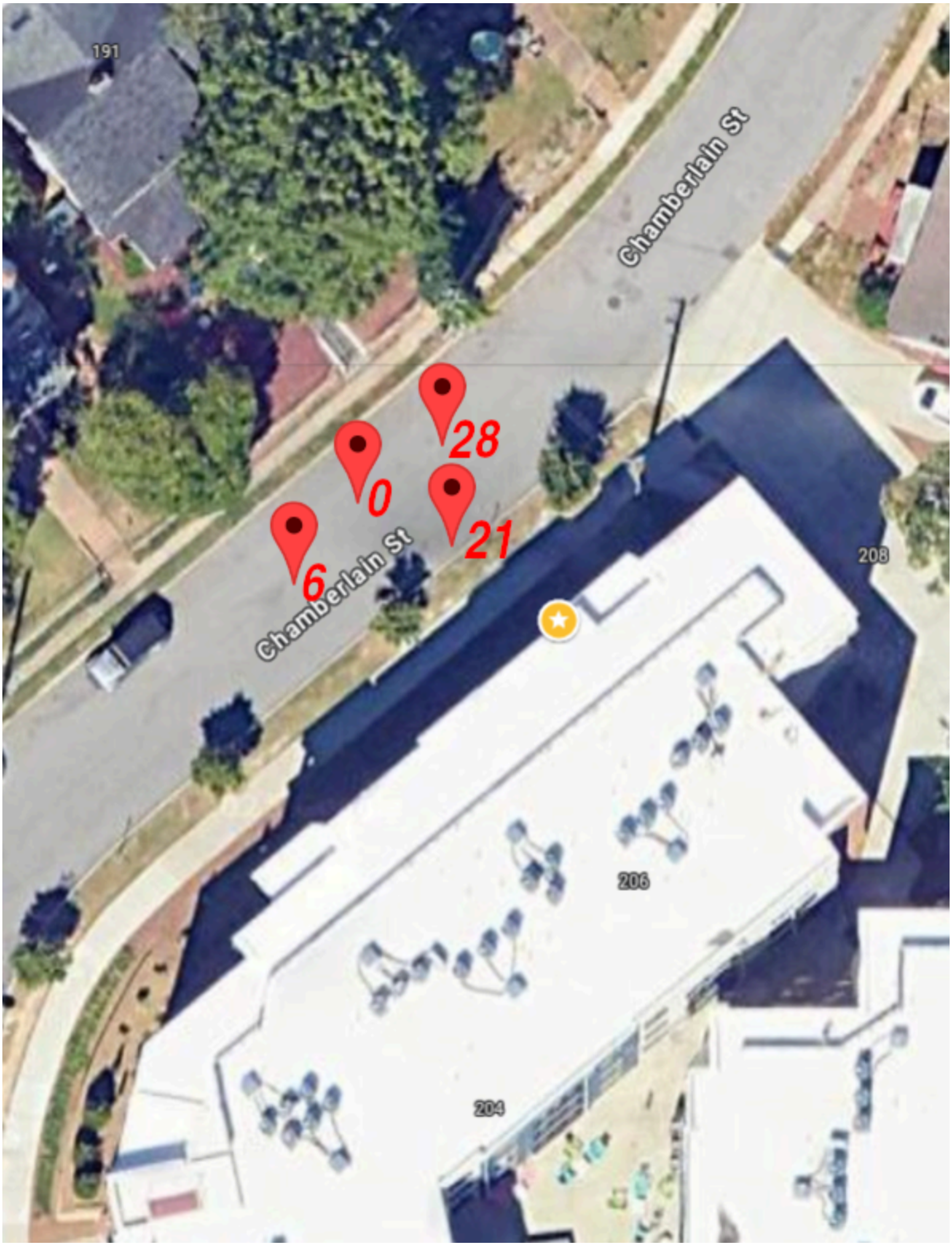

Figure 1: Selected points from test 1

where the mean value seems to be roughly twenty feet from my exact position. I believe this is because my GPS module only had a clear line of sight to northwestern satellites, which could have introduced some error in its triangulation calculations. To ensure that this is the case, I needed to repeat this test with a clearer view of the sky.

Test 2

The second test was run outside in an unobstructed area on a clear day, thus minimizing process error. For this test, we take our true state to be

$$x_{true} = \begin{bmatrix} 35.7866935 \\ 78.6666856 \\ 0 \\ 0 \end{bmatrix}$$

noting that the second entry is in *absolute* degrees, as the implementation considers all coordinates as positive degrees within a hemisphere.

The results for this test were very interesting due to a bug in the code that set the value of $\hat{x}_0 = 0$, which placed our initial guess ~300 miles south of the coast of Ghana. Nonetheless, the predicted position approached my true position rapidly:

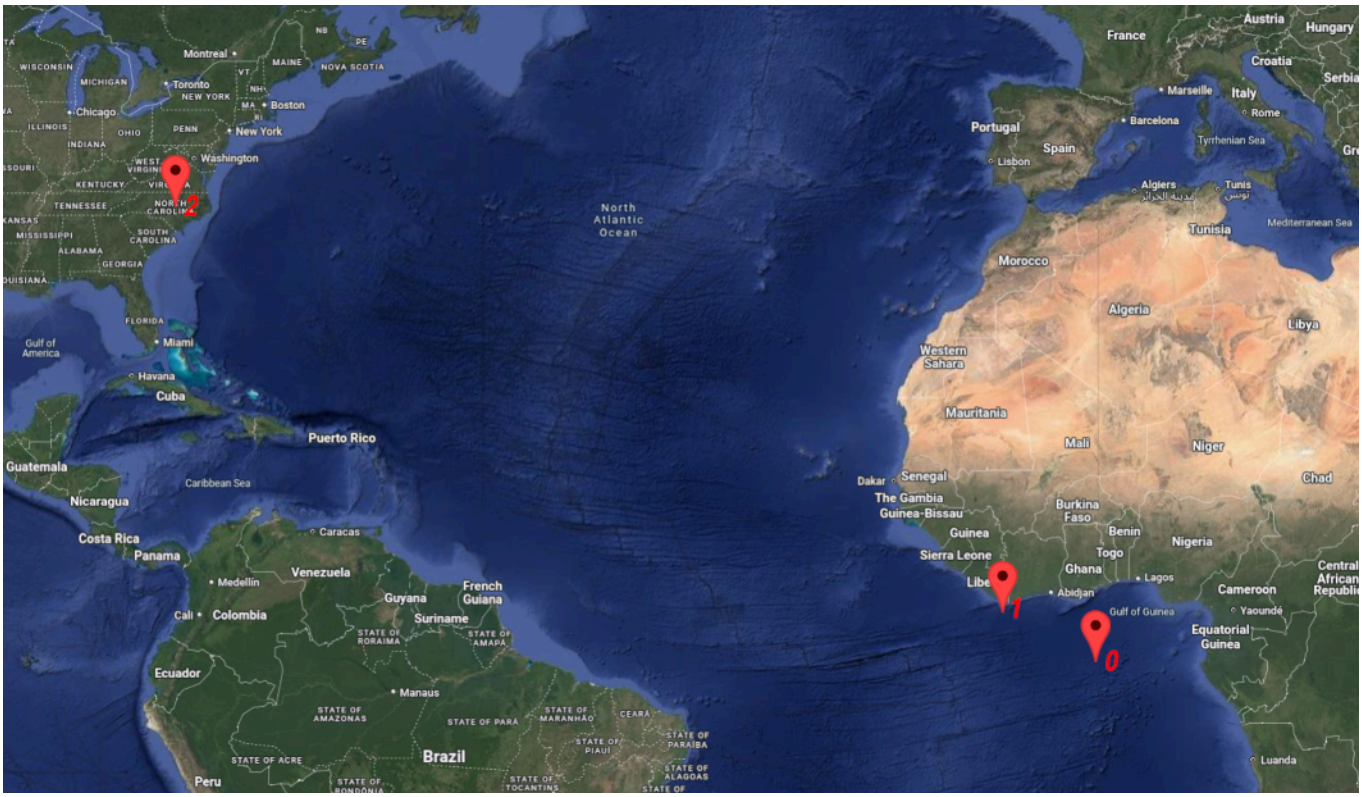

Figure 2: First two iterations of test 2

We can see from *Figure 2* that the filter only took two iterations to get to the correct city, with $\hat{x}_{2,2}$ being roughly 1 kilometer away from my true position. Another interesting observation of note was that $\hat{x}_{1,1} = \begin{bmatrix} 3.5786712 & 7.8666654 & 0 & 0 \end{bmatrix}^T$, which is almost exactly an order of magnitude smaller than x_{true}.

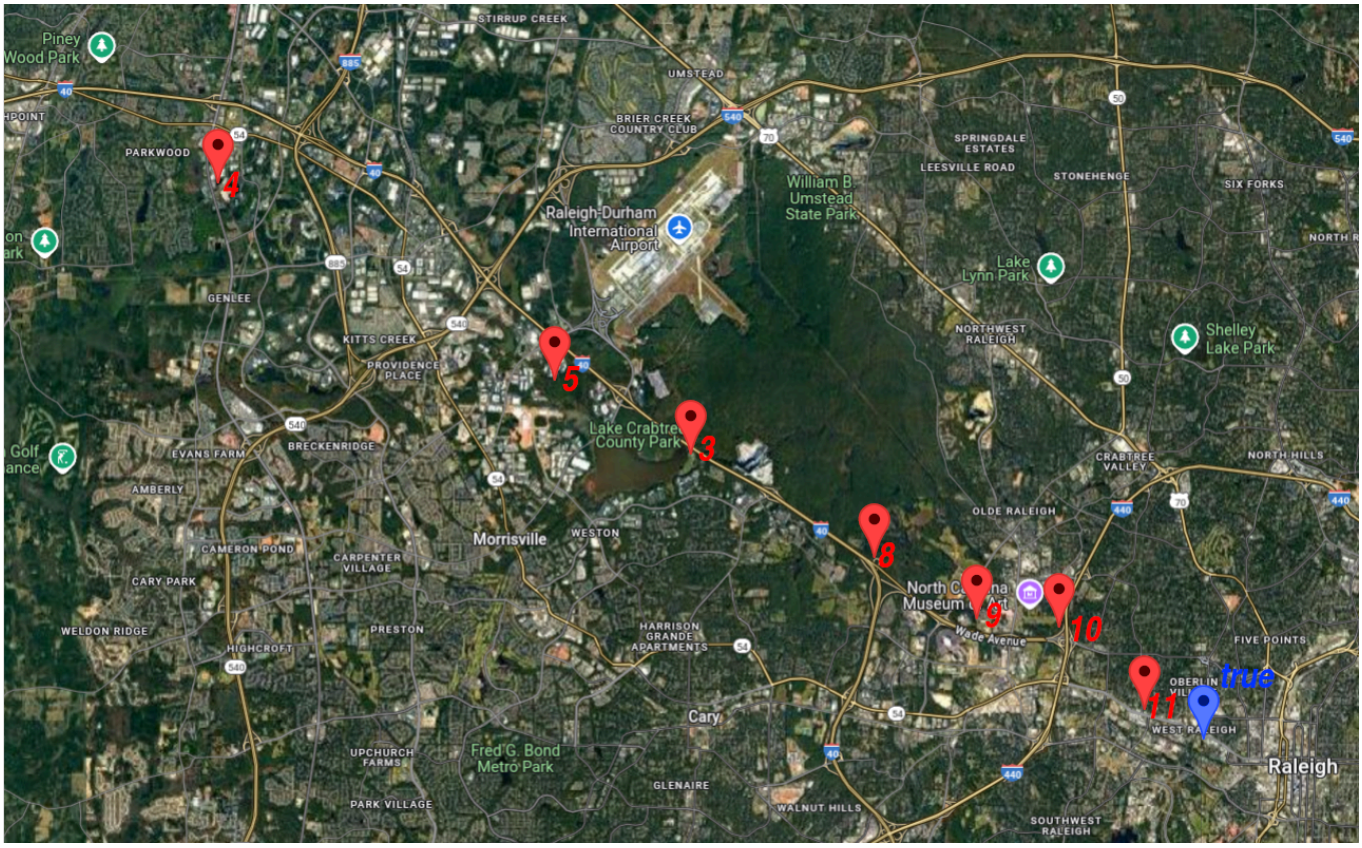

Figure 3: Overshoot and Correction on iterations 3 to 11

After a very close guess on the third iteration, the filter overshot my true position and ended up a few miles away. Interestingly, we see the estimates "turn around" and creep back towards our true position.

Figure 4: Overshoot on a Smaller Scale

We then see smaller oscillations as we approach nearer and nearer to the true point in *Figure 4*,

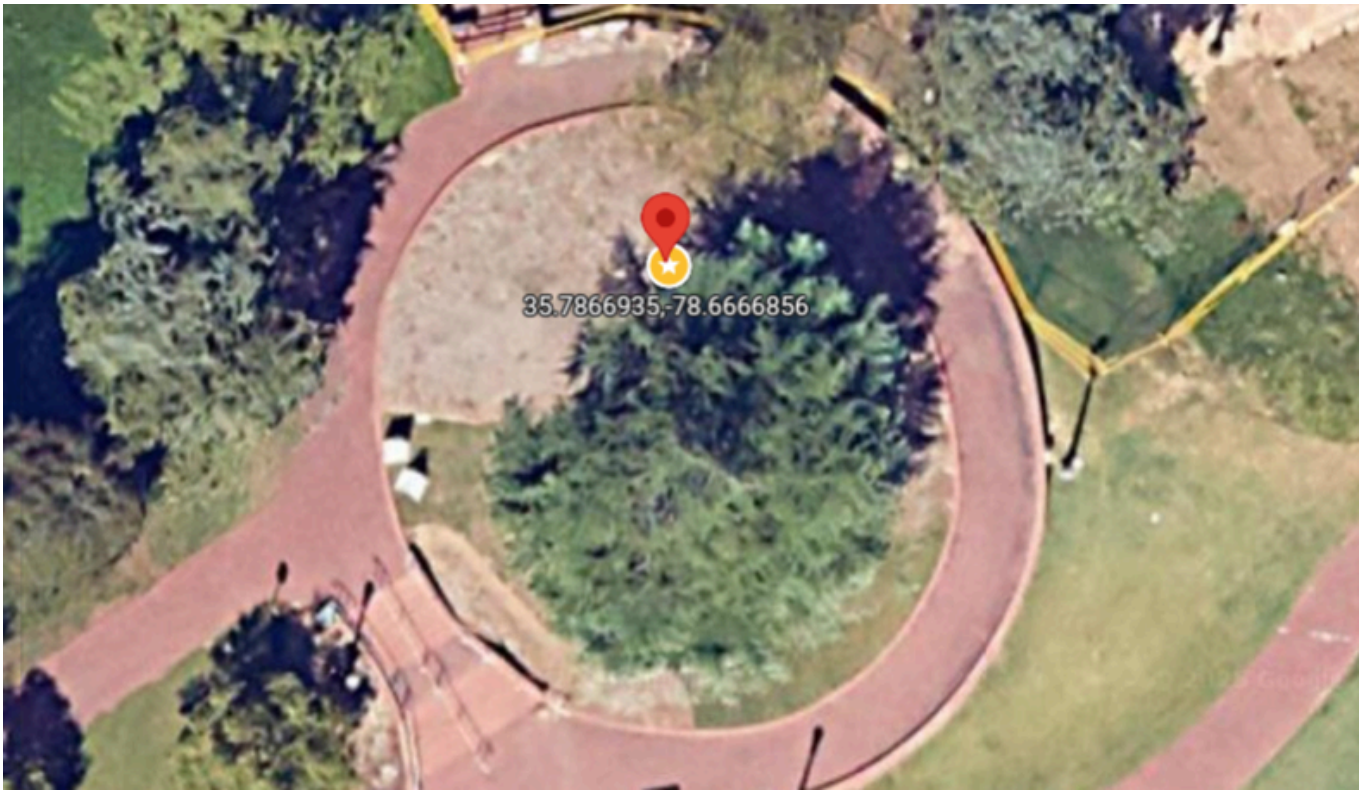

Figure 5: Convergence

until finally we converge to almost exactly to x_{true}. In this image the starred position is x_{true} and the red marker is the output of the fiftieth iteration $\hat{x}_{50,50}$. Taking the euclidean distance between the two points yields an error of **0.87 meters**, which is far below the 3 meters of precision that the manufacturer specifies.

Dynamic Testing

Due to time constraints and issues regarding the scope of this paper, the dynamic tests could not be conducted.

Interpretation

The erroneous $\hat{x}_{0,0} = 0$ initialization actually reveals a lot about the inner gears of the Kalman filter and highlights its place in control theory. In this section, we will explain why the filter seemed to repeatedly overshoot yet still settle to a highly accurate position.

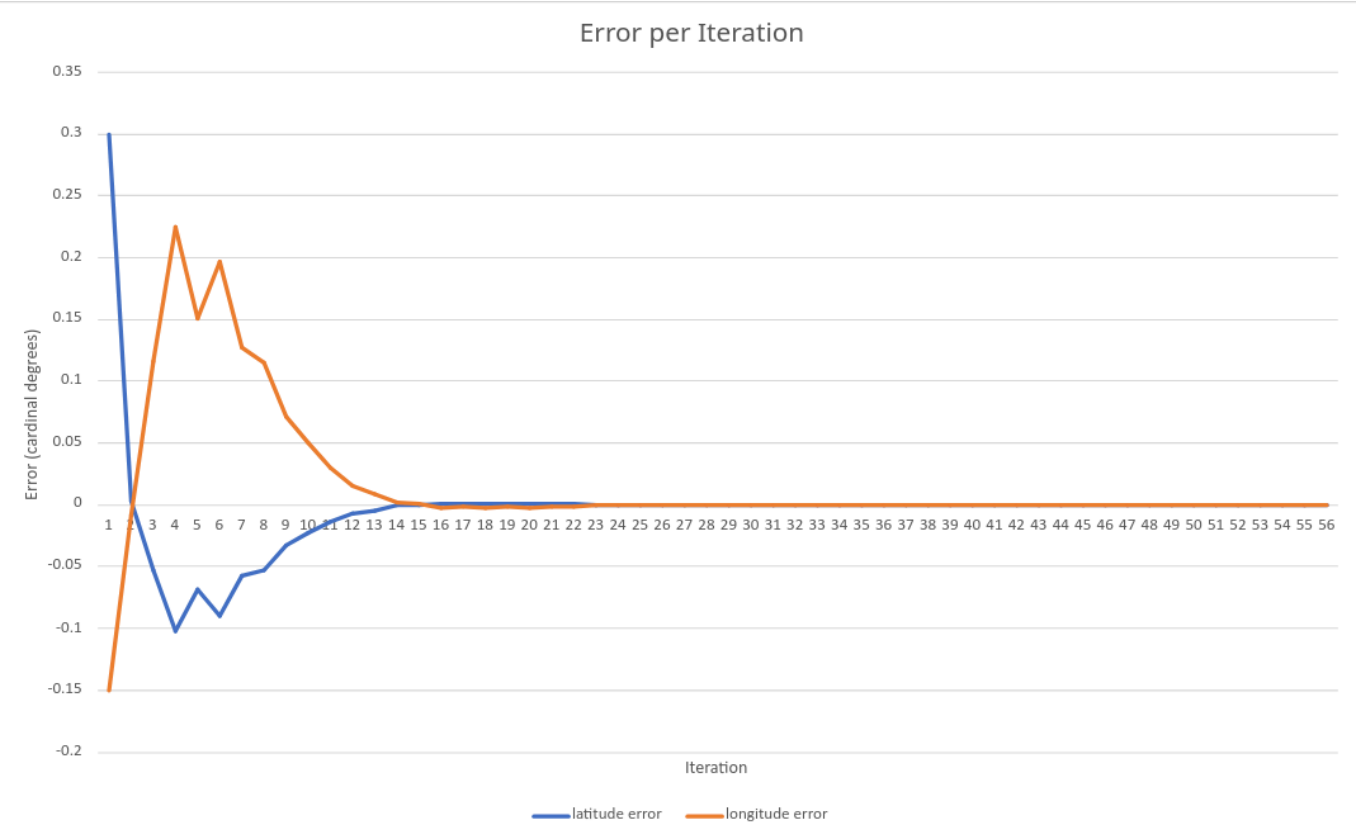

Figure 6: Error per Iteration. Error 0 omitted, Error 1 reduced for visualization purposes

Plotting our errors at each iteration shows a decaying oscillation pattern with a period of roughly 30 iterations, with a convergence at around the 25th iteration. This performance is impressive considering a new GPS measurement is available every 0.1 seconds, meaning even a terrible initial guess will converge to an accurate estimate within three seconds.

The fact that a Kalman filter shows this behavior was surprising at first, especially because the whole process seems linear in its construction, but when we view this from a control theorists perspective this behavior becomes obvious. A Kalman filter is a type of *observer*, which is essentially a controller that drives a state estimate rather than the state itself. A simpler kind of observer called the *Luenberger observer* is of a similar form to the Kalman filter's state estimation update equation, given by

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - C\hat{x})$$

where we observe a gain matrix L multiplied by the innovation $(y - C\hat{x})$ added to a prediction $A\hat{x} + Bu$ [6]. Observe the similarity to the Kalman filter's state update equation when we substitute in the previously extrapolated state for $\hat{x}_{n,n-1}$:

$$\hat{x}_{n,n} = [F\hat{x}_{n-1,n-1} + Gu_{n-1}] + K_n(z_n - H\hat{x}_{n,n-1})$$

we see a very similar gain matrix K_n multiplied by the innovation $(z_n - H\hat{x}_{n,n-1})$ added to an estimation of the state $F\hat{x}_{n-1,n-1} + Gu_{n-1}$.

The objective when using a Luenberger filter is to select L such that the eigenvalues of $(A - LC)$ are in a desirable place [6]. The desirable eigenvalues of $(A - LC)$ are often those in the open left-half plane, as these values will cause the observer error to approach 0 as $t \to \infty$. In essence, this is because we are working in the Laplace domain where time domain behavior is encoded through e^{-st}, therefore we want the exponential term to remain negative because

$$\lim_{t \to \infty} e^{(a+bi)t} = 0 \quad \forall a, b \in \mathbb{R} \mid a < 0$$

When we consider that a Kalman filter essentially chooses L for us, we can see how similar behavior should arise, though it would be more robust to show that K_n has eigenvalues in the open left-half plane for all n. The convergence to zero is due to a being negative, and the oscillations are due to the bi component, which Euler's identity tells us is equal to $\cos(b) + i * \sin(b)$.

Conclusion

The Kalman filter is an excellent tool for estimating true state from noisy measurement data. In this paper, we explored the rather simple implementation of a Kalman filter, and showed how rapidly it can recover from a poor initialization and converge to an estimation with low error. We also drew parallels to control theory in order to explain the decaying oscillation behavior observed when the filter overshoots. Observers like the Kalman filter are used in hundreds of systems in ways that we often take for granted, whether it's making GPS output more legible or fetching sensor data for a rocket ship, observers make our measurement tools safer and more reliable every day.

Works Cited

[1] Becker, Alex. Multivariate Kalman Filter. Retrieved from: https://www.kalmanfilter.net/multiSummary.html.
[2] Polotski, Vladimir. (2017). Re: Kalman filter, how do I choose initial P_0?. Retrieved from: https://www.researchgate.net/post/Kalman_filter_how_do_I_choose_initial_P_0/59e76b553d7f4badc12ffa2b/citation/download.
[3] Lacey, Tony. Tutorial: The Kalman Filter. Retrieved from: https://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf.
[4] Becker, Alex. The Kalman Gain. Retrieved from: https://www.kalmanfilter.net/kalmanGain.html
[5] Becker, Alex. Simplified Covariance Update Equation. Retrieved from: https://www.kalmanfilter.net/simpCovUpdate.html.
[6] Beard, Randal. Introduction to Feedback Control using Design Studies.

Appendix A: Kalman Filter Generic Implementation

main.cpp

```
#include <Arduino.h>
#include <chrono>
#include <mbed.h>
#include "gps.h"
#include "sensors.h"
#include <BasicLinearAlgebra.h>
#include "reckoner.h"


using namespace std::chrono_literals;
using namespace mbed;
using namespace rtos;

constexpr int n_z = 2;
constexpr int n_x = 4;
constexpr int n_u = 2;

const float deg_to_meter = 111320.0f; // meters per degree latitude

float latConversion;
float lonConversion;

Reckoner<n_x, n_z, n_u> reckoner;

void setup() {
    ThisThread::sleep_for(2s);
    IMU.begin();

    using namespace BLA;


    Matrix<n_x, n_x> processNoiseCovariance   = {1, 0, 0, 0,
                                                 0, 1, 0, 0,
                                                 0, 0, 1, 0,
                                                 0, 0, 0, 1};



    //control matrix (function of time)
    std::function<BLA::Matrix<n_x, n_u>(float)> controlMatrix = [](float
deltaT) {
        //delta t squared
        const float dt2 = deltaT * deltaT;
        BLA::Matrix<n_x, n_u> ret = {
```

```
            0.5 * dt2,  0,
            0,          0.5 * dt2,
            deltaT,     0,
            0,          deltaT
        };
        return ret;
    };

    //state transition matrix (function of time)
    std::function<BLA::Matrix<n_x, n_x>(float)> stateTransitionMatrix = []
(float deltaT) {
        BLA::Matrix<n_x, n_x> ret = {
            1, 0, deltaT, 0,
            0, 1, 0, deltaT,
            0, 0, 1, 0,
            0, 0, 0, 1
        };
        return ret;
    };

    // initialize H with
    std::unique_ptr<GpsData> data = nullptr;

    while(data == nullptr) {
        Serial.println("no fix yet");
        data = GPS::getInstance().listen();
        delay(100000);
    }
    Serial.println("got fix!");

    latConversion = deg_to_meter;
    lonConversion = deg_to_meter * cos(data->coord.latitude.degrees *
DEG_TO_RAD);

    float latInit = deg_to_meter * data->coord.latitude.degrees;
    float lonInit = lonConversion * data->coord.longitude.degrees ; //
longitude conversion is based on our latitude (small near poles, large near
equator)

    Matrix<n_z, n_x> obsMatrix = {
        1 / latConversion, 0, 0, 0,
        0,   1 / lonConversion, 0, 0
    };

    Matrix<n_x> x_init = {
        latInit,
```

```
        lonInit
    };

    Matrix<n_x, n_x> P_init = {
        9.0f / (latConversion * latConversion), 0, 0, 0,
        0, 9.0f / (lonConversion * lonConversion), 0, 0,
        0, 0, 1, 0,
        0, 0, 0, 1
    };
    Matrix<n_z, n_z> measurementCovariance = {
        9.0f / (latConversion * latConversion), 0,
        0, 9.0f / (lonConversion * lonConversion)
    };


    reckoner = Reckoner<n_x, n_z, n_u>(x_init, P_init,
stateTransitionMatrix, controlMatrix, obsMatrix, measurementCovariance,
processNoiseCovariance);
}

BLA::Matrix<n_u> getDirectionalAccel() {
    //TODO implement this in a future paper
    BLA::Matrix<n_u> ret = {
        0,
        0
    };
    return ret;
}

BLA::Matrix<n_z> coordToVec(coordinatePair &c) {
    BLA::Matrix<n_z> ret = {
        c.latitude.degrees,
        c.longitude.degrees
    };
    return ret;
}

void loop() {
    using namespace BLA;
    auto u = getDirectionalAccel();
    auto x_predicted = reckoner.predict(u);
    Serial.println("-------------------------------------------------");
    Serial.println();
    Serial.println("A priori state:");
    float lat = x_predicted(0) / latConversion;
    float lon = x_predicted(1) / lonConversion;
```

```
    Serial.print(lat, 7); Serial.print(" N, ");
    Serial.print(lon, 7); Serial.println(" W");
    Serial.println();
    std::unique_ptr<GpsData> gpsData = nullptr;
    while(gpsData == nullptr) {
        gpsData = GPS::getInstance().listen();
        delay(10);
    }

    auto z = coordToVec(gpsData->coord);
    Serial.println("Measurement:");
    Serial.print(z(0), 7); Serial.print(" N, ");
    Serial.print(z(1), 7); Serial.println(" W");
    Serial.println();
    Serial.println("A posteriori state:");
    auto x_estimate = reckoner.update(z);
    lat = x_estimate(0) / latConversion;
    lon = x_estimate(1) / lonConversion;
    Serial.print(lat, 7); Serial.print(" N, ");
    Serial.print(lon, 7); Serial.println(" W");


}
```

reckoner.h

```
/**
 * @file reckoner.h
 *
 * Provides interface for the reckoner class, an implementation of a Kalman
filter for approximating position
 */
#pragma once
#include <BasicLinearAlgebra.h>
#include <functional>

template<int n_x, int n_z, int n_u>
class Reckoner {
private:

    //state vectors
    BLA::Matrix<n_x> x_predict;
    BLA::Matrix<n_x> x_current;
    BLA::Matrix<n_x> x_past;
```

```
    //state transition matrix (as a function of time)
    std::function<BLA::Matrix<n_x, n_x>(float deltaT)> F;

    //control matrix (as a function of time)
    std::function<BLA::Matrix<n_x, n_u>(float deltaT)> G;

    //covariance estimate
    BLA::Matrix<n_x, n_x> P_predict;
    BLA::Matrix<n_x, n_x> P_current;
    BLA::Matrix<n_x, n_x> P_past;

    //process noise covariance
    BLA::Matrix<n_x, n_x> Q;

    //Measurement covariance
    BLA::Matrix<n_z, n_z> R;

    //process noise vector
    BLA::Matrix<n_x> w;

    //measurement noise vector
    BLA::Matrix<n_z> v;

    // Observation matrix
    BLA::Matrix<n_z, n_x> H;

    //Kalman Gain
    BLA::Matrix<n_x, n_z> K;

    BLA::Matrix<n_x, n_x> I_x;

    /**
     * extrapolates the state given control input and elapsed time
     * @param u control input
     * @param deltaT time since last call (in seconds)
     */
    BLA::Matrix<n_x> extrapolate_state(BLA::Matrix<n_u> u, float deltaT);

    /**
     * extrapolates state covariance given elapsed time
     * @param deltaT time since last call (in seconds)
     */
    BLA::Matrix<n_x, n_x> extrapolate_covariance(float deltaT);
    /**
     * updates the current estimate of the state given a measurement
     * @param z observed measurement
```

```
     */
    BLA::Matrix<n_x> update_state(BLA::Matrix<n_z> z);
    /**
     * updates the current estimate of the covariance
     */
    BLA::Matrix<n_x, n_x> update_covariance();
    /**
     * updates the Kalman gain
     */
    BLA::Matrix<n_x, n_z> update_K();
    /**
     * transforms a measurement in state space into one in measurement space
     */
    BLA::Matrix<n_z> transform_measurement(BLA::Matrix<n_x> measurement);

public:
    Reckoner(
        BLA::Matrix<n_x> &                                  x_init,
        BLA::Matrix<n_x, n_x> &                             P_init,
        std::function<BLA::Matrix<n_x, n_x>(float deltaT)>
state_transition,
        std::function<BLA::Matrix<n_x, n_u>(float deltaT)>  control_matrix,
        BLA::Matrix<n_z, n_x> &
observation_matrix,
        BLA::Matrix<n_z, n_z> &
measurement_covariance,
        BLA::Matrix<n_x, n_x> &
process_noise_covariance ) {

        I_x.Fill(0);
        for(int i = 0; i < n_x; i++) {
            I_x(i, i) = 1;
        }
        x_predict = x_init;
        x_current = x_init;
        x_past = x_init;
        F = state_transition;
        G = control_matrix;
        P_predict = P_init;
        P_current = P_init;
        P_past = P_init;
        w.Fill(0);
        v.Fill(0);
        H = observation_matrix;
        K.Fill(0);
        R = measurement_covariance;
```

```
        Q = process_noise_covariance;
    }

    Reckoner() {
    }

    /**
     * the update phase of the kalman filtering algorithm
     * @param z measurement taken of our system
     * @returns current estimate of our state
     */
    BLA::Matrix<n_x> update(BLA::Matrix<n_z> z);
    /**
     * the prediction phase of the kalman filtering algorithm
     * @param u control input to the system
     * @returns extrapolated estimate of our state
     */
    BLA::Matrix<n_x> predict(BLA::Matrix<n_u> u);
};

template<int n_x, int n_z, int n_u>
BLA::Matrix<n_x> Reckoner<n_x, n_z, n_u>::update(BLA::Matrix<n_z> z) {
    using namespace BLA;
    x_past = x_predict;
    P_past = P_predict;
    // TODO i think this is the problem
    K = update_K();
    x_current = update_state(z);

    P_current = update_covariance();

    return x_current;
}

template<int n_x, int n_z, int n_u>
BLA::Matrix<n_x> Reckoner<n_x, n_z, n_u>::predict(BLA::Matrix<n_u> u) {
    static unsigned long last;
    static float deltaT;
    unsigned long now = millis();

    if(!last || last > millis()) {
        last = 10;
        //leave deltaT as is

    } else {
        deltaT = static_cast<float>(now - last);
```

```
        last = now;
        //convert milliseconds to seconds
        deltaT = deltaT / 1000;
    }
    x_predict = extrapolate_state(u, deltaT);
    P_predict = extrapolate_covariance(deltaT);

    return x_predict;
}



// Private methods
template<int n_x, int n_z, int n_u>
BLA::Matrix<n_x> Reckoner<n_x, n_z, n_u>::extrapolate_state(BLA::Matrix<n_u>
u, float deltaT) {
    return F(deltaT) * x_current + G(deltaT) * u;
}

template<int n_x, int n_z, int n_u>
BLA::Matrix<n_x, n_x> Reckoner<n_x, n_z, n_u>::extrapolate_covariance(float
deltaT) {
    return F(deltaT) * P_current * ~(F(deltaT)) + Q;
}

template<int n_x, int n_z, int n_u>
BLA::Matrix<n_x> Reckoner<n_x, n_z, n_u>::update_state(BLA::Matrix<n_z> z) {
    return x_past + K * (z - H * x_past);
}

template<int n_x, int n_z, int n_u>
BLA::Matrix<n_x, n_x> Reckoner<n_x, n_z, n_u>::update_covariance() {
    BLA::Matrix<n_x, n_x> T = (I_x - (K * H));
    return T * P_past * ~T + K * R * ~K;
}

template<int n_x, int n_z, int n_u>
BLA::Matrix<n_x, n_z> Reckoner<n_x, n_z, n_u>::update_K() {
    return P_past * ~H * BLA::Inverse(H * P_past * ~H + R);
}

template<int n_x, int n_z, int n_u>
BLA::Matrix<n_z> Reckoner<n_x, n_z,
n_u>::transform_measurement(BLA::Matrix<n_x> measurement) {
    return H * measurement;
}
```

Appendix B: Serial Output

Note: only the first 57 measurements are shown

```
-----------------------------------------------

A priori state:
0.0000000 N, 0.0000000 W

$GPGGA,171116.000,3547.2024,N,07839.9993,W,1,06,1.51,119.8,M,-33.0,M,,*5E
$GPRMC,171116.000,A,3547.2024,N,07839.9993,W,0.04,0.00,200425,,,A*76
$GPGGA,171117.000,3547.2024,N,07839.9993,W,1,06,1.51,119.9,M,-33.0,M,,*5E
$GPRMC,171117.000,A,3547.2024,N,0783Measurement:
35.7867088 N, 78.6666489 W

A posteriori state:
3.5786712 N, 7.8666654 W
-----------------------------------------------

A priori state:
3.5786712 N, 7.8666654 W

$GPGGA,171256.000,3547.2015,N,07839.9999,W,2,07,1.36,125.2,M,-33.0,M,,*57
$GPRMC,171256.000,A,3547.2015,N,07839.9999,W,0.07,0.00,200425,,,D*7F
Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.7831841 N, 78.6589584 W
-----------------------------------------------

A priori state:
35.9022598 N, 78.9207077 W

$GPGGA,171257.000,3547.2015,N,07839.9999,W,2,07,1.36,125.3,M,-33.0,M,,*57
$GPRMC,17Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8396721 N, 78.7831268 W
```

```
-------------------------------------------------

A priori state:
35.9819603 N, 79.0959015 W

1257.000,A,3547.2015,N,07839.9999,W,0.04,0.00,200425,,,D*7D
Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8886414 N, 78.8907776 W
-------------------------------------------------

A priori state:
35.8968925 N, 78.9089050 W

$GPGGA,171258.000,3547.2014,N,07839.9999,W,2,07,1.36,125.3,M,-33.0,M,,*59
$GPRMeasurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8552895 N, 78.8174515 W
-------------------------------------------------

A priori state:
35.9665413 N, 79.0620117 W

MC,171258.000,A,3547.2014,N,07839.9999,W,0.03,0.00,200425,,,D*74
Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8763275 N, 78.8637085 W
-------------------------------------------------

A priori state:
35.8820572 N, 78.8763046 W

$GPGGA,171259.000,3547.2013,N,07839.9998,W,2,07,1.36,125.4,M,-33.0,M,,*59
$GPRMeasurement:
```

```
35.7866898 N, 78.6666641 W

A posteriori state:
35.8445244 N, 78.7937927 W
-------------------------------------------------

A priori state:
35.9022217 N, 78.9206314 W

MC,171259.000,A,3547.2013,N,07839.9998,W,0.02,0.00,200425,,,D*72
Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8392067 N, 78.7821045 W
-------------------------------------------------

A priori state:
35.8415451 N, 78.7872467 W

$GPGGA,171300.000,3547.2013,N,07839.9998,W,2,07,1.36,125.5,M,-33.0,M,,*55
$GPRMeasurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8189697 N, 78.7376175 W
-------------------------------------------------

A priori state:
35.8388176 N, 78.7812500 W

MC,171300.000,A,3547.2013,N,07839.9998,W,0.01,0.00,200425,,,D*7C
Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8097382 N, 78.7173309 W
-------------------------------------------------

A priori state:
```

```
35.8102646 N, 78.7184906 W

$GPGGA,171301.000,3547.2012,N,07839.9998,W,2,07,1.36,125.6,M,-33.0,M,,*56
$Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.8004608 N, 78.6969376 W
-------------------------------------------------

A priori state:
35.8031693 N, 78.7028809 W

GPRMC,171301.000,A,3547.2012,N,07839.9998,W,0.01,0.00,200425,,,D*7C
Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.7939606 N, 78.6826477 W
-------------------------------------------------

A priori state:
35.7938423 N, 78.6823807 W

$GPGGA,171302.000,3547.2012,N,07839.9998,W,2,07,1.36,125.6,M,-33.0,M,,*55
$GPRMC,1Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
35.7908669 N, 78.6758423 W
-------------------------------------------------

A priori state:
35.7882195 N, 78.6700211 W

71302.000,A,3547.2012,N,07839.9998,W,0.01,0.00,200425,,,D*7F
Measurement:
35.7866898 N, 78.6666641 W

A posteriori state:
```

```
35.7873650 N, 78.6681442 W
------------------------------------------------

A priori state:
35.7871208 N, 78.6676102 W

$GPGGA,171303.000,3547.2013,N,07840.0000,W,2,07,1.36,125.7,M,-33.0,M,,*5B
$GPRMC,17Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7869415 N, 78.6672211 W
------------------------------------------------

A priori state:
35.7838020 N, 78.6603317 W

1303.000,A,3547.2013,N,07840.0000,W,0.02,0.00,200425,,,D*73
Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7854195 N, 78.6638870 W
------------------------------------------------

A priori state:
35.7852554 N, 78.6635208 W

$GPGGA,171304.000,3547.2013,N,07840.0000,W,2,07,1.36,125.7,M,-33.0,M,,*5C
$GPRMC,1Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7858505 N, 78.6648331 W
------------------------------------------------

A priori state:
35.7837105 N, 78.6601334 W

71304.000,A,3547.2013,N,07840.0000,W,0.02,0.00,200425,,,D*74
```

```
Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7853699 N, 78.6637802 W
------------------------------------------------

A priori state:
35.7852592 N, 78.6635437 W

$GPGGA,171305.000,3547.2013,N,07840.0001,W,2,07,1.36,125.8,M,-33.0,M,,*53
$GPRMC,17Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7858543 N, 78.6648407 W
------------------------------------------------

A priori state:
35.7846832 N, 78.6622696 W

1305.000,A,3547.2013,N,07840.0001,W,0.02,0.00,200425,,,D*74
Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7858086 N, 78.6647415 W
------------------------------------------------

A priori state:
35.7857628 N, 78.6646423 W

$GPGGA,171306.000,3547.2013,N,07840.0003,W,2,07,1.36,125.8,M,-33.0,M,,*52
$GPRMC,17Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7861481 N, 78.6654892 W
------------------------------------------------
```

```
A priori state:
35.7856445 N, 78.6643753 W

1306.000,A,3547.2013,N,07840.0003,W,0.03,0.00,200425,,,D*74
Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7862320 N, 78.6656570 W
------------------------------------------------

A priori state:
35.7862167 N, 78.6656189 W

$GPGGA,171307.000,3547.2014,N,07840.0004,W,2,07,1.36,125.8,M,-33.0,M,,*53
$GPRMMeasurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7864113 N, 78.6660538 W
------------------------------------------------

A priori state:
35.7862549 N, 78.6657028 W

C,171307.000,A,3547.2014,N,07840.0004,W,0.03,0.00,200425,,,D*75
Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7864952 N, 78.6662445 W
------------------------------------------------

A priori state:
35.7864914 N, 78.6662292 W

$GPGGA,171308.000,3547.2015,N,07840.0004,W,2,07,1.36,125.9,M,-33.0,M,,*5C
$GPRMC,Measurement:
35.7866898 N, 78.6666718 W
```

```
A posteriori state:
35.7865753 N, 78.6664124 W
-----------------------------------------------

A priori state:
35.7865601 N, 78.6663818 W

171308.000,A,3547.2015,N,07840.0004,W,0.02,0.00,200425,,,D*7A
Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7866325 N, 78.6665497 W
-----------------------------------------------

A priori state:
35.7866364 N, 78.6665497 W

$GPGGA,171309.000,3547.2015,N,07840.0005,W,2,07,1.36,125.9,M,-33.0,M,,*5C
$GMeasurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7866592 N, 78.6666031 W
-----------------------------------------------

A priori state:
35.7866859 N, 78.6666641 W

PRMC,171309.000,A,3547.2015,N,07840.0005,W,0.02,0.00,200425,,,D*7A
Measurement:
35.7866898 N, 78.6666718 W

A posteriori state:
35.7866898 N, 78.6666641 W
-----------------------------------------------

A priori state:
35.7866898 N, 78.6666718 W
```

```
$GPGGA,171310.000,3547.2015,N,07840.0006,W,2,07,1.36,125.9,M,-33.0,M,,*57
$GPRMC,Measurement:
35.7866898 N, 78.6666794 W

A posteriori state:
35.7866898 N, 78.6666718 W
-------------------------------------------------

A priori state:
35.7867203 N, 78.6667328 W

171310.000,A,3547.2015,N,07840.0006,W,0.01,0.00,200425,,,D*72
Measurement:
35.7866898 N, 78.6666794 W

A posteriori state:
35.7867012 N, 78.6667023 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6667023 W

$GPGGA,171311.000,3547.2015,N,07840.0006,W,2,07,1.36,125.9,M,-33.0,M,,*56
$Measurement:
35.7866898 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666870 W
-------------------------------------------------

A priori state:
35.7867126 N, 78.6667328 W

GPRMC,171311.000,A,3547.2015,N,07840.0006,W,0.01,0.00,200425,,,D*73
Measurement:
35.7866898 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6667023 W
-------------------------------------------------
```

```
A priori state:
35.7866974 N, 78.6667023 W

$GPGGA,171312.000,3547.2016,N,07840.0008,W,2,07,1.36,125.9,M,-33.0,M,,*58
$GPRMC,Measurement:
35.7866936 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666870 W
-------------------------------------------------

A priori state:
35.7867088 N, 78.6667175 W

171312.000,A,3547.2016,N,07840.0008,W,0.01,0.00,200425,,,D*7D
Measurement:
35.7866936 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6667023 W
-------------------------------------------------

A priori state:
35.7866974 N, 78.6667023 W

$GPGGA,171313.000,3547.2017,N,07840.0008,W,2,07,1.36,125.9,M,-33.0,M,,*58
$GPRMMeasurement:
35.7866974 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666870 W
-------------------------------------------------

A priori state:
35.7867050 N, 78.6667023 W

C,171313.000,A,3547.2017,N,07840.0008,W,0.01,0.00,200425,,,D*7D
Measurement:
35.7866974 N, 78.6666794 W
```

```
A posteriori state:
35.7867012 N, 78.6666870 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666870 W

$GPGGA,171314.000,3547.2017,N,07840.0009,W,2,07,1.36,125.9,M,-33.0,M,,*5E
$GPMeasurement:
35.7866974 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666794 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666794 W

RMC,171314.000,A,3547.2017,N,07840.0009,W,0.01,0.00,200425,,,D*7B
Measurement:
35.7866974 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666794 W
-------------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

$GPGGA,171315.000,3547.2018,N,07840.0009,W,2,07,1.36,125.9,M,-33.0,M,,*50
$GPRMC,Measurement:
35.7866974 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666794 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666794 W
```

```
171315.000,A,3547.2018,N,07840.0009,W,0.01,0.00,200425,,,D*75
Measurement:
35.7866974 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666794 W
-------------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

$GPGGA,171316.000,3547.2018,N,07840.0009,W,2,07,1.36,125.9,M,-33.0,M,,*53
$GPRMeasurement:
35.7866974 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666794 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666794 W

MC,171316.000,A,3547.2018,N,07840.0009,W,0.01,0.00,200425,,,D*76
Measurement:
35.7866974 N, 78.6666794 W

A posteriori state:
35.7866974 N, 78.6666794 W
-------------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

$GPGGA,171317.000,3547.2019,N,07840.0010,W,2,07,1.36,125.9,M,-33.0,M,,*5B
$GPMeasurement:
35.7867012 N, 78.6666870 W

A posteriori state:
35.7867012 N, 78.6666794 W
```

```
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666870 W

RMC,171317.000,A,3547.2019,N,07840.0010,W,0.02,0.00,200425,,,D*7D
Measurement:
35.7867012 N, 78.6666870 W

A posteriori state:
35.7867012 N, 78.6666870 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666870 W

$GPGGA,171318.000,3547.2019,N,07840.0010,W,2,07,1.36,125.9,M,-33.0,M,,*54
$GPRMeasurement:
35.7867012 N, 78.6666870 W

A posteriori state:
35.7867012 N, 78.6666870 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666870 W

MC,171318.000,A,3547.2019,N,07840.0010,W,0.02,0.00,200425,,,D*72
Measurement:
35.7867012 N, 78.6666870 W

A posteriori state:
35.7867012 N, 78.6666870 W
-------------------------------------------------

A priori state:
35.7867012 N, 78.6666870 W

$GPGGA,171319.000,3547.2019,N,07840.0010,W,2,07,1.36,125.8,M,-33.0,M,,*54
$Measurement:
```

```
35.7867012 N, 78.6666870 W

A posteriori state:
35.7867012 N, 78.6666870 W
------------------------------------------------

A priori state:
35.7867012 N, 78.6666870 W

GPRMC,171319.000,A,3547.2019,N,07840.0010,W,0.02,0.00,200425,,,D*73
Measurement:
35.7867012 N, 78.6666870 W

A posteriori state:
35.7867012 N, 78.6666870 W
------------------------------------------------

A priori state:
35.7867012 N, 78.6666870 W

$GPGGA,171320.000,3547.2018,N,07840.0010,W,2,07,1.36,125.8,M,-33.0,M,,*5F
$GPRMeasurement:
35.7866974 N, 78.6666870 W

A posteriori state:
35.7866974 N, 78.6666870 W
------------------------------------------------

A priori state:
35.7866974 N, 78.6666870 W

MC,171320.000,A,3547.2018,N,07840.0010,W,0.01,0.00,200425,,,D*7B
Measurement:
35.7866974 N, 78.6666870 W

A posteriori state:
35.7866974 N, 78.6666870 W
------------------------------------------------

A priori state:
```

```
35.7866974 N, 78.6666870 W

$GPGGA,171321.000,3547.2018,N,07840.0010,W,2,07,1.36,125.8,M,-33.0,M,,*5E
$GPRMeasurement:
35.7866974 N, 78.6666870 W

A posteriori state:
35.7866974 N, 78.6666870 W
-----------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

MC,171321.000,A,3547.2018,N,07840.0010,W,0.01,0.00,200425,,,D*7A
Measurement:
35.7866974 N, 78.6666870 W

A posteriori state:
35.7866974 N, 78.6666794 W
-----------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

$GPGGA,171322.000,3547.2018,N,07840.0010,W,2,07,1.36,125.7,M,-33.0,M,,*52
$GPMeasurement:
35.7866974 N, 78.6666870 W

A posteriori state:
35.7866974 N, 78.6666794 W
-----------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

RMC,171322.000,A,3547.2018,N,07840.0010,W,0.01,0.00,200425,,,D*79
Measurement:
35.7866974 N, 78.6666870 W

A posteriori state:
```

```
35.7866974 N, 78.6666794 W
------------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

$GPGGA,171323.000,3547.2018,N,07840.0010,W,2,07,1.36,125.7,M,-33.0,M,,*53
$GPRMC,Measurement:
35.7866974 N, 78.6666870 W

A posteriori state:
35.7866974 N, 78.6666794 W
------------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W

171323.000,A,3547.2018,N,07840.0010,W,0.00,0.00,200425,,,D*79
Measurement:
35.7866974 N, 78.6666870 W

A posteriori state:
35.7866974 N, 78.6666794 W
------------------------------------------------

A priori state:
35.7866974 N, 78.6666794 W
```