UAV State Estimation Using IMU Vector with DNN.

Group 7

ITCS 5156

By: Rohit Alavala, Saiphani Jasthi, Yuehan Lan, Christian Micklisch, Sai Nihanth Vanam

Problem Statement (Brief introduction)

While there exist state estimators for Unmanned Aerial Vehicles, UAVs, in simulated environments few rely on strictly the raw data produced by inertial measurement units, IMUs. IMU's represent a mixture of sensors utilized to help determine the linear accelerations, angular velocities and heading. The linear accelerations are provided by the accelerometer, the angular velocities are handled by the gyroscope, and the heading is handled by a magnetometer. The state of a UAV contains the linear and angular positions, relative to where it started. The linear position is the x, y and z position along a 3D grid where x represents the distance, y represents depth, and z represents height. The angular position is the ϕ, Θ, and Ψ. The diagram to the right shows what the state would be for the linear and angular positions.

For state estimation there is also a subsection, attitude estimation, that focuses on only determining the angular positions, e.g. ϕ, Θ, and Ψ.

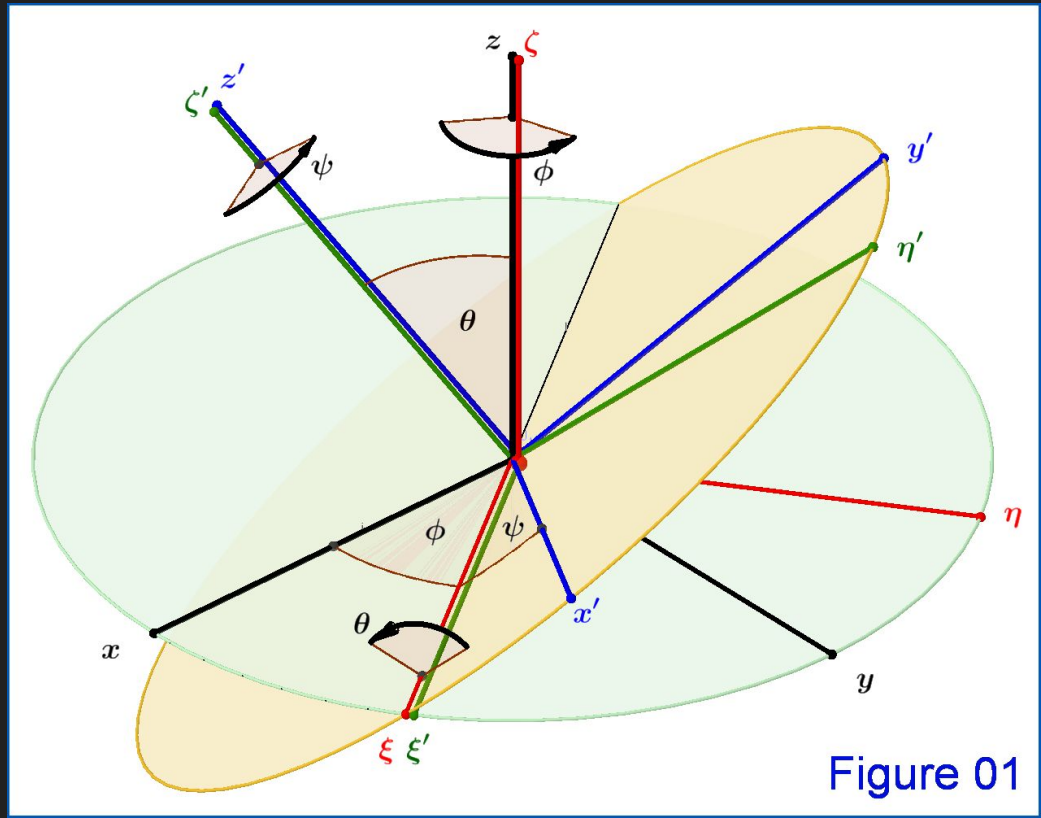


Figure 01

Problem Statement (The Problem)

The IMU outputs have a lot of sensor noise associated with it, which makes it difficult to write a set of equations that will describe the attitude of the UAV directly from the outputs of the accelerometer and gyroscope. Currently a Kalman Filter can be utilized to digitally filter out the noise of the sensors. The output of the filtered data can then help estimate the attitude of a UAV's position, which can then be utilized to estimate the state. This process of using a Kalman Filter to determine the state is called the Extended Kalman Filter. (Yang et al.)

A survey of literature of UAV state estimation shows that, most state estimators that utilize machine learning models, rely on on/off-board optical sensors to train the model for state estimation. Currently the only research on IMU sensor usage for UAVs are Al-Sharman et al., they created a self-written DLNN to replace the Kalman filter to help estimate the attitude, finding that the DLNN is more accurate than the Kalman Filter. The results of Al-Sharman et al.'s DLNN are shown below.

Our goal is to create a model that can estimate the state of the UAV from the raw data using a generalized model of the IMU error dynamics. This model would act as a Extended Kalman Filter for any IMU sensor.

State	KF RMSEE	DLNN RMSEE
ϕ	0.0017	0.00016
θ	0.0018	0.00023
ψ	0.0130	0.00200

Problem Statement (What are our challenges?)

When relying on IMU sensor data to train the model one issue is that it's dependent on the sensor we select. Each sensor has its own error dynamics wherein the noisy output may be directed towards a specific average value or the output has a certain gaussian noise. This means that, were we to train our model on a specific IMU, then the model might only work on the specific IMU. Another issue that represents itself is there is currently no data published by Al-Sharman et al. nor DLNN code, that can be utilized to help train a model. This produces 2 main challenges:

1. The sensor that we train the model on determines the sensor that we need to utilize when feeding data to the model
2. There is no data available to recreate Al-Sharman et al.'s model, let alone train our own model.

Problem Statement (How will we resolve the problem?)

To resolve both issues, we need to somehow create data that will reflect the real state of the UAV, while also providing the current IMU readings of the accelerometer and gyroscope. This data also needs to represent multiple different sensors, with a broader range of error dynamics that can reflect varying IMUs. The data also needs to represent multiple different flights to allow various error dynamics of the IMU to show themselves. To do this we utilize Micklisch et al.'s simulator to generate the data, which utilizes simulated sensors to comprise a IMU with broad error dynamics. The simulator can be run to output a CSV that represents a single flight path which will contain the real state of the UAV in the simulated environment along with the accelerometer and gyroscope readings of the IMU. This data is pulled during the flight every 0.03 seconds. Utilizing this simulator we can produce hundreds of random flight path CSVs which can then be preprocessed into a single CSV to train the model. (This data would be simulated and self-made)

The Simulator

Micklisch et al.'s simulator, used to produce the training data, was originally implemented to test various controlling algorithms utilized for autonomous quadrotors. For autonomous flight a quadrotor will determine autonomously how to reach a given position utilizing an takeoff/landing controller, attitude controller, and collision detection. The takeoff/landing controller helps determine the speed at which to start up/stop and the height at which to stabilize itself before heading towards the given position. The attitude controller determines the angle at which the quadrotor should be at, for it to move towards the given position. The obstacle detection method should prevent the quadrotor from crashing into a wall and direct the quadrotor around an obstacle to reach the given position safely. Each of the components that a quadrotor utilizes to move and detect were recreated inside of the simulator. The inertial structure of the quadrotor, motor control systems, propeller shape, are also variable, allowing any current production quadrotor to be recreated. The IMU, Ultrasonic, and pressure sensors are also variable allowing for multiple different error dynamics, inertial structures, and refresh rates to be set.

The Simulation Data

To run the simulator and generate various flights, a bash script was written to trigger the simulator to run and capture all of the printed data into a csv file. The image on the right describes all of the data files. Each "desired_linear_position_*" was randomly set in the beginning of the flight. Currently the specific Attitude Controller was set to Edward Kempers PID controller. White noise was used for the error dynamic of the IMU. This allows error dynamic to be as broad as possible with sensor readings fluctuating randomly given the quadrotors specific state. In total around 502 flights were generated.

Column Name	Description
timestamp	Refers to the amount of time that has passed since the start of the simulation. Time passes at indexes of 0.03 seconds.
attitude_controller_id	Refers to the attitude controller that is utilized during the flight. All flights should use the Kemper, "k", attitude controller. The attitude controller defines how the quadrotor is going to fly
desired_linear_position_x	The target position that the drone wants to be at in the x direction
desired_linear_position_y	The target position that the drone wants to be at in the y direction
desired_linear_position_z	The target position that the drone wants to be at in the z direction
linear_position_x	The actual x linear position of the drone in the simulated environment.
linear_position_y	The actual y linear position of the drone in the simulated environment.
linear_position_z	The actual z linear position of the drone in the simulated environment.
angular_position_phi	The actual ϕ angular position of the drone in the simulated environment.
angular_position_theta	The actual θ angular position of the drone in the simulated environment.
angular_position_psi	The actual ψ angular position of the drone in the simulated environment.
accelerometer_reading_x	The reading of the accelerometer attached to the drone from the x direction.
accelerometer_reading_y	The reading of the accelerometer attached to the drone from the y direction.
accelerometer_reading_z	The reading of the accelerometer attached to the drone from the z direction.
gyroscope_reading_phi	The reading of the gyroscope attached to the drone from the ϕ direction.
gyroscope_reading_theta	The reading of the gyroscope attached to the drone from the θ direction.
gyroscope_reading_psi	The reading of the gyroscope attached to the drone from the ψ direction.

Approach (Summary)

To train a model that will produce the same output as an EKF we need to generate several hundred flights in random directions by utilizing a pre-existing simulator (Micklisch et al.). Each flight will be recorded as a CSV. An input target pair table will be generated by randomly selecting a flight recording, f, and a moment in the flight, i, and getting the next, x, rows from i. These x rows represent a stream of raw IMU readings that will be stored as a 6 DOF vector, this vector is the input. The difference in angular and linear positions from i to i+x will be the target. This dataset will then be trained and tested with a DNN Regression model. The RMSEE will then be compared to Al-Sharman et al.'s results by comparing the numerical numbers of his RMSEE score.

We will provide several other regression metrics of our models to provide a better perspective of its performance, e.g. MSE, RMSE, MAE, …

How the data was Pre-processed

Since our data is scattered through 502 different files with several thousands of rows in each one, we want to create a dataframe that will describe the change in the x, y, z, ϕ, θ, and ψ from a vector of readings of the accelerometer and gyroscope. To get the vector information together, with the change of the angular/linear positions, a function gets the data of a random file, get a random sequential list of rows in the file, and get the difference in time and positions along with a vector of the readings starting at the beginning row.

To get a random file, we load all of the files in the "data" directory as a list, select a random file name from the list, and load the file as a pandas DataFrame. Each file has the CSV headers in the first line.

To get a random list of sequential rows we select a random starting position and determine the end position by adding the start position to the total amount of rows we want to pick. The list of rows from a random starting position (minus one) to the end position (starting position + amount). The difference of the quadrotors position needs to be calculated from at least 2 rows, as you need the first row to provide insight into the readings of the quadrotor (IMU readings) and the second row to see where the quadrotor ended up (linear and angular positions). The function to get a random list of rows is shown below.

```
def get_random_rows(df, amount):
    if (df.shape[0] - amount <= 1):
        return pd.DataFrame()
    start_index = random.randint(1, df.shape[0] - amount)
    return df.iloc[(start_index - 1):(amount + start_index), :]
```

How the data was Pre-processed (continued)

To get the difference in the start and end positions, along with a vector of the IMU readings, we utilize the randomly selected rows. These rows are a DataFrame, which allows us to get the difference in values from the first to the last element for all of the state positions and create the vector of all of the IMU readings by utilizing the "iloc" array positions. The code for converting the random rows selection to a single row for the training dataset is shown below.

```
def convert_to_table_row(df):
    try:
        row = [
            df["timestamp"].iloc[-1] - df["timestamp"].iloc[0], # time_change
            df["linear_position_x"].iloc[-1] - df["linear_position_x"].iloc[0], # x_change
            df["linear_position_y"].iloc[-1] - df["linear_position_y"].iloc[0], # y_change
            df["linear_position_z"].iloc[-1] - df["linear_position_z"].iloc[0], # z_change
            df["angular_position_phi"].iloc[-1] - df["angular_position_phi"].iloc[0], # phi_change
            df["angular_position_theta"].iloc[-1] - df["angular_position_theta"].iloc[0], # theta_change
            df["angular_position_psi"].iloc[-1] - df["angular_position_psi"].iloc[0] # psi_change
        ];

        row += df["accelerometer_reading_x"].iloc[:-1].tolist()
        row += df["accelerometer_reading_y"].iloc[:-1].tolist()
        row += df["accelerometer_reading_z"].iloc[:-1].tolist()
        row += df["gyroscope_reading_phi"].iloc[:-1].tolist()
        row += df["gyroscope_reading_theta"].iloc[:-1].tolist()
        row += df["gyroscope_reading_psi"].iloc[:-1].tolist()
        return row
    except IndexError as error:
        print(df + " failed");
        return False
```

How the data was Pre-processed (continued)

To create our training table we simply determine the amount of randomly selected rows and the amount of rows we want in our training table. We then select a random file and get the rows until our table is filled. The code to generate the training table is shown below.

```
def generate_table(files, vector_size, row_size):
    df_first = get_dataframe(files[0]);
    columns = df_first.columns
    table = []

    for i in range(row_size):
        random_rows = get_random_rows(read_random_file(files, columns), vector_size)
        if (random_rows.empty):
            continue

        row = convert_to_table_row(random_rows)
        if (row == False):
            continue
        table.append(row)

    return pd.DataFrame(table, columns=generate_columns(vector_size))
```

Our training table will then have a structure shown on the right, where the vector size is two. Several of these files are then generated to allow the model to train against various row amounts and vector sizes. This is done for experimentation purposes, although a vector size of 900 is excessive, it can still provide insight into how the model trains against our data structure. The next slide shows all of the vector sizes and row amounts created. All of the data is stored in a google drive folder.

Column Names	Description
time_change	The amount of time that has passed overall
x_change	The change in the x position.
y_change	The change in the y position.
z_change	The change in the z position.
phi_change	The change in the ϕ angular position since the start of the flight
theta_change	The change in the θ angular position since the start of the flight
psi_change	The change in the ψ angular position since the start of the flight
accelerometer_reading_x_0	The first x reading from the accelerometer.
accelerometer_reading_x_1	The last x reading from the accelerometer.
accelerometer_reading_y_0	The first y reading from the accelerometer.
accelerometer_reading_y_1	The last y reading from the accelerometer.
accelerometer_reading_z_0	The first z reading from the accelerometer.
accelerometer_reading_z_1	The last z reading from the accelerometer.
gyroscope_reading_phi_0	The first ϕ reading from the gyroscope.
gyroscope_reading_phi_1	The last ϕ reading from the gyroscope.
gyroscope_reading_theta_0	The first θ reading from the gyroscope.
gyroscope_reading_theta_1	The last θ reading from the gyroscope.
gyroscope_reading_psi_0	The first ψ reading from the gyroscope.
gyroscope_reading_psi_1	The last ψ reading from the gyroscope.

Code to generate each Training Table

```
PROCESSED_FILE_PATHS = '../processed_data/'

def create_files(files, vector_sizes, row_sizes):
    for vector_size in vector_sizes:
        for row_size in row_sizes:
            generate_table(files, vector_size, row_size).to_csv(
                PROCESSED_FILE_PATHS +
                "row_size" + str(vector_size) +
                "_vector_size" + str(row_size) + ".csv", index=False, header=True);

create_files(files, [1, 2, 10, 25, 100, 225], [2000, 4000, 6000, 8000, 10000]);
```

DNN Inputs and Outputs

- Inputs:
  - All(depends on dataset) Accelerometer_reading_x,y,z
  - All(depends on dataset) Gyroscope_reading_phi,theta,psi
- Outputs/Targets(Current Focus):
  - phi_change
  - theta_change
  - psi_change
  - x_change
  - y_change
  - z_change

About the Neural Networks

We are using the Keras Sequential model.

The sequential model uses 5 Dense layers. Each layer uses ‘sigmoid’ activation. The first layer has 256 nodes and every layer afterwards uses half the amount of the previous layers nodes and the last layer uses the number of the targets for node count. Using Stochastic gradient descent(SGD) as the optimizer. Some hyperparameters are ‘epochs’, ‘batch size’, ‘validation split’, ‘learning rate’, ‘optimizer’, and ‘activation’. We learned that predicting one target at a time gave the best results. Each state value(target value) will be predicted using its own model.

We also explored the normal Keras model but received poorer results.

Sequential Neural Network Structure and Results

Structure:

```
model = Sequential()
model.add(Dense(256, input_dim=in_dim, activation="sigmoid"))
model.add(Dense(128, activation="sigmoid"))
model.add(Dense(64, activation="sigmoid"))
model.add(Dense(32, activation="sigmoid"))
model.add(Dense(out_dim))
#model.compile(loss="mse", optimizer="adam")
model.compile(loss="mse", optimizer="sgd")
```

Training:

```
model.fit(X_train, t_train, epochs=200, batch_size=12, verbose=2, validation_split=0.4)
```

Sequential Neural Network Structure and Results

- Results:

phi_change

```
1 ypred = model.predict(X_test)
2 print("y1 RMSE: ", (np.sqrt(mean_squared_error(t_test.iloc[:, 0], ypred[:,0]))))
```

```
y1 RMSE:  0.002973207765195292
```

theta_change

```
1 ypred = model.predict(X_test)
2 print("y1 RMSE: ", (np.sqrt(mean_squared_error(t_test.iloc[:, 0], ypred[:,0]))))
```

```
y1 RMSE:  0.0022240176776540353
```

psi_change

```
1 ypred = model.predict(X_test)
2 print("y1 RMSE: ", (np.sqrt(mean_squared_error(t_test.iloc[:, 0], ypred[:,0]))))
```

```
y1 RMSE:  0.0034372640167258032
```

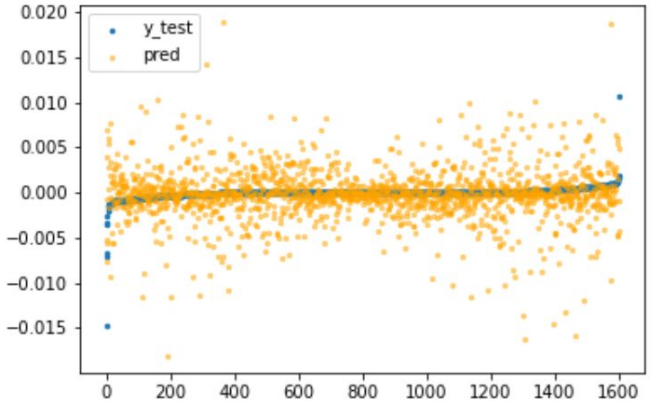
Sequential Predictions vs Target Plot for given structure

Predicting phi_change

```
x_ax = range(len(X_test))

y_test_index = np.argsort(t_test.iloc[:, 0], axis=0).to_numpy()

plt.scatter(x_ax, t_test.iloc[y_test_index],  s=6, label="y_test")
plt.scatter(x_ax, ypred[y_test_index], s=6, label="pred",c="orange", alpha=0.5)
#plt.ylim(t_test.iloc[y_test_index[0]].to_numpy()[0])
plt.legend()
plt.show()
```

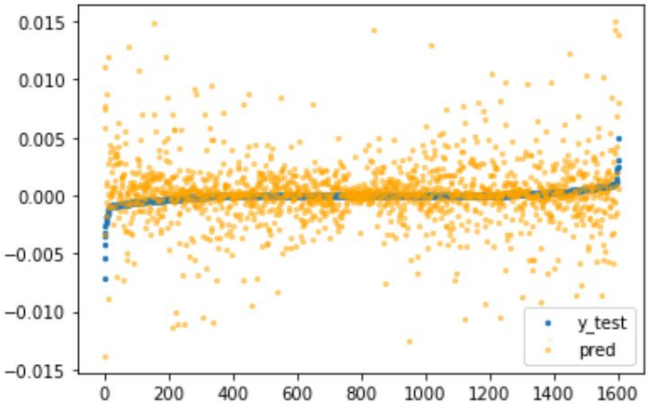


Predicting theta_change

```
x_ax = range(len(X_test))

y_test_index = np.argsort(t_test.iloc[:, 0], axis=0).to_numpy()

plt.scatter(x_ax, t_test.iloc[y_test_index],  s=6, label="y_test")
plt.scatter(x_ax, ypred[y_test_index], s=6, label="pred",c="orange", alpha=0.5)
#plt.ylim(t_test.iloc[y_test_index[0]].to_numpy()[0])
plt.legend()
plt.show()
```

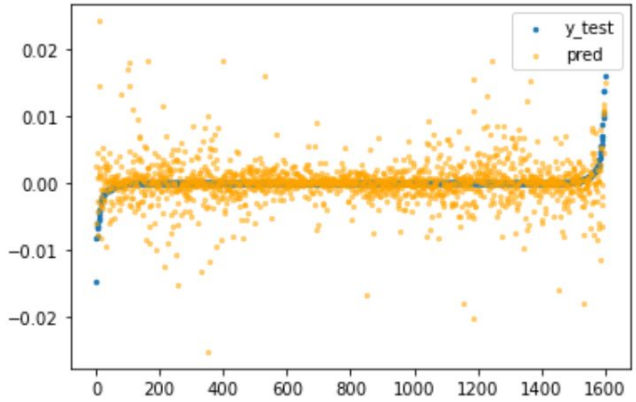


Predicting psi_change

```
x_ax = range(len(X_test))

y_test_index = np.argsort(t_test.iloc[:, 0], axis=0).to_numpy()

plt.scatter(x_ax, t_test.iloc[y_test_index],  s=6, label="y_test")
plt.scatter(x_ax, ypred[y_test_index], s=6, label="pred",c="orange", alpha=0.5)
#plt.ylim(t_test.iloc[y_test_index[0]].to_numpy()[0])
plt.legend()
plt.show()
```



Normal Neural Network Structure and Results

Input features are accelerometer reading of x, y, z, phi, theta, psi.

Target is the change of phi.

Structure:

```
input_layer = Input(shape=(X.shape[1],))
dense_layer_1 = Dense(100, activation='relu')(input_layer)
dense_layer_2 = Dense(50, activation='relu')(dense_layer_1)
dense_layer_3 = Dense(25, activation='relu')(dense_layer_2)
output = Dense(1)(dense_layer_3)

model = Model(inputs=input_layer, outputs=output)
model.compile(loss="mean_squared_error" , optimizer="adam", metrics=["mean_squared_error"])
```

Training:

```
history = model.fit(X_train, y_train, batch_size=2, epochs=100, verbose=1, validation_split=0.2)
```

Normal Neural Network Structure and Results for Phi

```
pred_train = model.predict(X_train)
print('MSE:',(mean_squared_error(y_train,pred_train)))
print('RMSE:', np.sqrt(mean_squared_error(y_train,pred_train)))

pred = model.predict(X_test)
print('MSE:', (mean_squared_error(y_test,pred)))
print('RMSE:', np.sqrt(mean_squared_error(y_test,pred)))
```

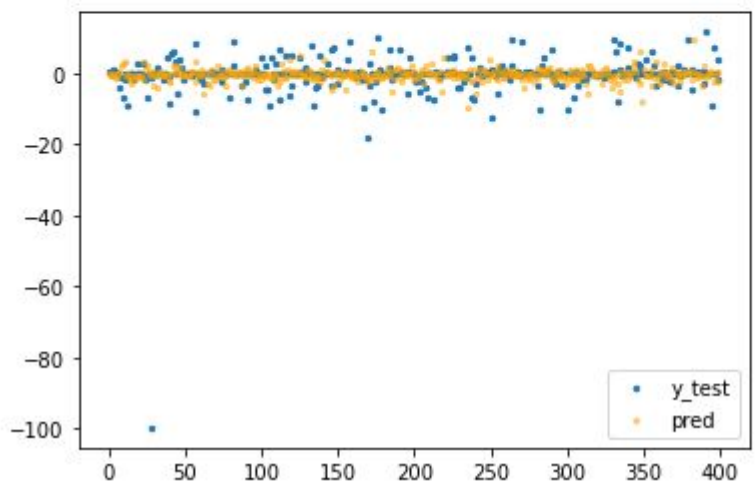
```
MSE: 17.485722447093707
RMSE: 4.181593290492716
MSE: 35.61123388566977
RMSE: 5.967514883573377
```

Predictions vs Target Plot for given structure

```
x_ax = range(len(X_test))

plt.scatter(x_ax, y_test,  s=6, label="y_test")
plt.scatter(x_ax, pred, s=6, label="pred",c="orange", alpha=0.5)

plt.legend()
plt.show()
```



Future plans for NN structures

We plan on making a class that will take in each of our pre-processed data files and loop through many combinations of hyperparameter values and saves a plot for each experiment to a file. This will allow us to try many combinations of differently pre-processed datasets and many different combinations of hyperparameters while staying organized. Once we are satisfied with the number of experiments we can look through all the output plots to pick the best NN structure for our task.

We also want to focus on the Sequential Keras model as it gave the best results.

Accomplished and Incomplete Milestones

Accomplished	Never Ending	Incomplete
Survey of literature on DNN usage in determining attitude, pose and state estimation for unmanned vehicles	Compare Results from Al-Sharman et al.	Create a model that estimates the state (linear and angular positions)
Configure Simulator to output real state and IMU readings	Experiment with varying layer sizes and structures of the vector sizes of the data, DNN layers and hyperparameters.	Create a class that can produce multiple different graphs to represent the real position of the simulation vs. the state estimated.
Pre-process the data, structure multiple train		Create a class to produce multiple different metrics for each of the models, e.g. MSE, RMSE, MAE, ...
Research on which libraries to utilize for DNN Regression		Create a "RunModel" class that fits models, plots their predictions, and generates metrics for various vector sizes, layer structures, and hyperparameters.,
Create models with different libraries that estimates the attitude (angular position)		Create a notebook that describes how each of the classes is built and their purpose.
Make MidTerm Report		

Accomplished and Incomplete Milestones

Accomplished	Never Ending	Incomplete
		Create a notebook to utilize the "RunModel" class and experiment with varying layer sizes and structures of the vector sizes of the data, DNN layers and hyperparameters.
		Write up Final Report
		Present the Final Report Slides

Difficulties or Problems that you are experiencing

- One difficulty we are having is finding the optimal NN structure. But we do have an idea of how to tackle the problem as mentions in “Future plans for NN structure” slide.

Plan / Tasks	Rohit Alavala	Saiphani Jasthi	Yuehan Lan	Christian Micklisch	Target Date
Survey of literature on DNN usage in determining attitude, pose and state estimation for unmanned vehicles	x		x		02/12/21
Configure Simulator to output real state and IMU readings		x	x		02/13/21
Adjust simulator to alter dataset output for various different flights	x			x	02/14/21
Pre-process the data, structure multiple train		x	x		03/14/21
Research on which libraries to utilize for DNN Regression		x		x	03/14/21
Create a model that estimates the attitude (angular position) using three models.		x		x	03/15/21
Compare RMSEE Results from Al-Sharman et al.'s paper.	x				03/16/21
Describe the preprocessed data, and how the simulator works in the Report				x	03/17/21
Write MidTerm Report	x	x	x	x	03/18/21
Create a model that estimates the state (linear and angular positions) using six separate models for each position.		x		x	03/20/21

Plan **Tasks (Continued)**	Rohit Alavala	Saiphani Jasthi	Yuehan Lan	Christian Micklisch	Target Date
Create a class that can produce multiple different graphs to represent the real position of the simulation vs. the state estimated.	x		x		03/22/21
Create a class to produce multiple different metrics for each of the models, e.g. MSE, RMSE, MAE, ...		x	x		03/23/21
Create a "RunModel" class that fits models, plots their predictions, and generates metrics for various vector sizes, layer structures, and hyperparameters.,	x			x	03/24/21
Create a notebook that describes the different how each of the classes is built and their purpose.		x	x		03/25/21
Create a notebook to utilize the "RunModel" class and experiment with varying layer sizes and structures of the vector sizes of the data, DNN layers and hyperparameters.		x		x	03/26/21
Write up Final Report	x	x	x	x	04/15/21
Present the Final Report Slides	x	x	x	x	04/29/21

References

- Al-Sharman, Mohammad K., et al. "Deep-learning-based neural network training for state estimation enhancement: Application to attitude estimation." IEEE Transactions on Instrumentation and Measurement 69.1 (2019): 24-34.
- Aznar, Fidel, Mar Pujol, and Ramón Rizo. "Visual navigation for uav with map references using convnets." Conference of the Spanish Association for Artificial Intelligence. Springer, Cham, 2016.
- Carrio, Adrian, et al. "A review of deep learning methods and applications for unmanned aerial vehicles." Journal of Sensors 2017 (2017).
- Cheon, Kangbeom, et al. "On replacing PID controller with deep learning controller for DC motor system." Journal of Automation and Control Engineering Vol 3.6 (2015).
- Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.
- Ibarra-Bonilla, Mariana N., P. Jorge Escamilla-Ambrosio, and Juan Manuel Ramirez-Cortes. "Attitude estimation using a Neuro-Fuzzy tuning based adaptive Kalman filter." Journal of Intelligent & Fuzzy Systems 29.2 (2015): 479-488.
- Jaradat, Mohammad Abdel Kareem, and Mamoun F. Abdel-Hafez. "Non-linear autoregressive delay-dependent INS/GPS navigation system using neural networks." IEEE sensors journal 17.4 (2016): 1105-1115.
- Jin, Ren, et al. "Drone detection and pose estimation using relational graph networks." Sensors 19.6 (2019): 1479.
- Kemper E. Quad-rotor flight path energy optimization. Doctoral Dissertation, Northern Arizona University.
- Micklisch, Christian, Nathan Hilliard, and Hala ElAarag. "Modeling and simulation of autonomous quadrotor systems and their onboard sensors." SIMULATION (2020): 0037549720974753.
- Yang, Hyunsoo, et al. "Multi-rotor drone tutorial: systems, mechanics, control and state estimation." Intelligent Service Robotics 10.2 (2017): 79-93.

Proposal Slides

UAV State Estimation Using IMU Vector with DNN.

Group 7

ITCS 5156

By: Rohit Alavala, Saiphani Jasthi, Yuehan Lan, Christian Micklisch, Sai Nihanth Vanam

Problem Statement

While there exist state estimators for Unmanned Aerial Vehicles, UAVs, in simulated environments few rely on strictly the raw data produced by inertial measurement units, IMUs. Most estimators that utilize machine learning models rely on, on/off-board optical sensors, and digital filters to train the models on state estimation. Our implementation aims to estimate the state by providing raw IMU data to a DNN with simulated data, using Micklisch et al.'s simulator, of 501 random quadrotor flights.

Motivation

One of the most difficult areas of designing a UAV is state estimation, especially with common off-the-shelf components. Current commercial IMU's are cheaply available but contain sensor error dynamics such as noise and drift. These error dynamics are usually resolved by the use of an extended Kalman filter (EKF), reviewed by Yang et al., but are complex to implement and require background knowledge of the Inertial body of the UAV. Al-Sharman et al. replaced the Kalman filter and fed the raw IMU data directly to the ML model. This resulted in an RMSEE score for the angular positions phi, theta, and psi of 0.00016, 0.00023, and 0.00200 respectively. We want to see if we can improve the accuracy of this model by sending the raw IMU data as a vector of recorded readings over a time period and predict not only the attitude, angular, position but the linear position as well.

Literature Survey - Summary of Related Approaches

As described by Yang et al. traditional State Estimation can be done using three core techniques, attitude estimation via an Inertial Measurement Unit (IMU), Pose estimation via extended Kalman filtering, and optimal state estimation. Outside of traditional works in state estimation Machine learning methods have been derived. Carrio et al. touched on the use of DNN and CNN for motion control with low-level control policies from imperfect optical sensor data in simulations implemented by Aznar et al. and Ross Girshick respectively. Jaradat and Hafez explored an integrated navigation solution, INS, that uses IMU sensors to provide data to a Neural Network which has been pre-trained with the sensors' error dynamics. This INS system characterizes the x, y, and z vector spaces individually. The INS was initially tested in a simulated environment and then run against real GPS/IMU data. Cheon et al. utilized Proportional Integral Derivative, PID, controller outputs to train a Deep Learning Controller on how to control a DC Motor System. Ibarra-Bonilla et al. developed a Kalman Filter with Neuro-Fuzzy logic for attitude estimation using the raw data of an IMU. Jin et al. proposed a relational graph network derived from a relational neural network to accurately predict a 6D pose of a quadrotor from optical sensors. Al-Sharman et al. implemented a deep learning framework to improve state estimation by identifying measurement noise and filtering it out.

Literature Survey - Criticize The Existing Approaches

Ibarra-Bonilla et. al's attitude estimation was only utilized with Pedestrian Dead Reckoning, PDR, and didn't compare the Neuro-Fuzzy adaptation of the Kalman Filter with the current UAV flight. The use of state estimation from optical sensors by Jin et al., Aznar et al., and Ross Girshick may be ineffective in foggy or low light situations. Jaradat and Hafez's use of GPS fusion will limit their Navigation System in Urban and indoor environments.

Literature Survey - What Are The Pros and Cons?

Pro: A deep neural network (DNN) can ensure identifying representative, fitted, and robust models for the unfiltered measurement noise. The deep-learning-based neural network (DLNN) attitude estimation results exhibit superiority over the conventional attitude estimates.

Con: The usage of DNN and DLNN models may only learn the error dynamics of the sensors it was trained against, limiting the models robustness against other sensors.

Literature Survey - How Do You Relate Them To Your Planned Approach

I-Sharman et al.'s approach to comparing the RMSEE scores of the DLNN model and the Kalman filter is appropriate for determining how well the DLNN compares to the Kalman filter introduced by Yang et al. We would like to adopt this approach by comparing the results of an Extended Kalman Filter to our DLNN model.

Approach (Summary)

To train a model that will produce the same output as an EKF we need to generate several hundred flights in random directions by utilizing a pre-existing simulator (Micklisch et al.). Each flight will be recorded as a CSV. An input target pair table will be generated by randomly selecting a flight recording, f, and a moment in the flight, i, and getting the next, x, rows from i. These x rows represent a stream of raw IMU readings that will be stored as a 6 DOF vector, this vector is the input. The difference in angular and linear positions from i to i+x will be the target. This dataset will then be trained and tested with a DNN Regression model. The RMSEE will then be compared to Al-Sharman et al.'s results.

Questions We Want To Answer From The Project

- Can we replace an extended Kalman filter with an ML model that can estimate the attitude of the quadrotor from a stream of raw IMU data?
- How large does a vector of IMU data need to be to accurately determine the state? Can a vector of size 1 with the accelerometer and gyroscope readings be enough or do we need at least 20 readings, 0.6 seconds of flight, to predict the linear and angular location?

Expectations of What We Will Be Able To Learn

- Data Preprocessing
- Preprocess timestamped simulation data into a vector format.
- Learn how to apply DNN Regression to a Vector of IMU data
- Learn Fast R-CNN for object detection

Plan / Tasks	Rohit Alavala	Saiphani Jasthi	Yuehan Lan	Christian Micklisch	Target Date
Survey of literature on DNN usage in determining attitude, pose and state estimation for unmanned vehicles	x		x		02/12/21
Configure Simulator to output real state and IMU readings		x	x		02/13/21
Adjust simulator to alter dataset output for various different flights	x			x	02/14/21
Pre-process the data, structure multiple train		x	x		03/14/21
Research on which libraries to utilize for DNN Regression		x		x	03/14/21
Create a model that estimates the attitude (angular position)		x		x	03/15/21
Compare Results from Al-Sharman et al.	x				03/16/21
Write MidTerm Report	x	x	x	x	03/18/21
Create a model that estimates the state (linear and angular positions)		x		x	03/25/21
Experiment with varying layer sizes and structures	x		x		03/30/21
Write up Final Report	x	x	x	x	04/15/21
Present the Final Report Slides	x	x	x	x	04/29/21

References

- Al-Sharman, Mohammad K., et al. "Deep-learning-based neural network training for state estimation enhancement: Application to attitude estimation." IEEE Transactions on Instrumentation and Measurement 69.1 (2019): 24-34.
- Aznar, Fidel, Mar Pujol, and Ramón Rizo. "Visual navigation for uav with map references using convnets." Conference of the Spanish Association for Artificial Intelligence. Springer, Cham, 2016.
- Carrio, Adrian, et al. "A review of deep learning methods and applications for unmanned aerial vehicles." Journal of Sensors 2017 (2017).
- Cheon, Kangbeom, et al. "On replacing PID controller with deep learning controller for DC motor system." Journal of Automation and Control Engineering Vol 3.6 (2015).
- Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.
- Ibarra-Bonilla, Mariana N., P. Jorge Escamilla-Ambrosio, and Juan Manuel Ramirez-Cortes. "Attitude estimation using a Neuro-Fuzzy tuning based adaptive Kalman filter." Journal of Intelligent & Fuzzy Systems 29.2 (2015): 479-488.
- Micklisch, Christian, Nathan Hilliard, and Hala ElAarag. "Modeling and simulation of autonomous quadrotor systems and their onboard sensors." SIMULATION (2020): 0037549720974753.
- Jaradat, Mohammad Abdel Kareem, and Mamoun F. Abdel-Hafez. "Non-linear autoregressive delay-dependent INS/GPS navigation system using neural networks." IEEE sensors journal 17.4 (2016): 1105-1115.
- Jin, Ren, et al. "Drone detection and pose estimation using relational graph networks." Sensors 19.6 (2019): 1479.
- Yang, Hyunsoo, et al. "Multi-rotor drone tutorial: systems, mechanics, control and state estimation." Intelligent Service Robotics 10.2 (2017): 79-93.