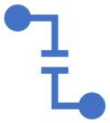
AMEND ML

4/1/22

Problem Statement

Main Goal: Track satellites to maximize Gain-to-Noise

Develop controls **to** physically move ground dishes to point in proper direction



Normally controls are done with traditionally calibrated PID systems

Now implementing with machine **learning** techniques based on simulation databases

Project Description

We want to design and implement a ML algorithm to accept a wide range of RF inputs and generate PI parameters to take predictive actions.

The goal is to create increasingly accurate PI parameters that produce improved corrective actions for the tracking system.

Main steps to take:

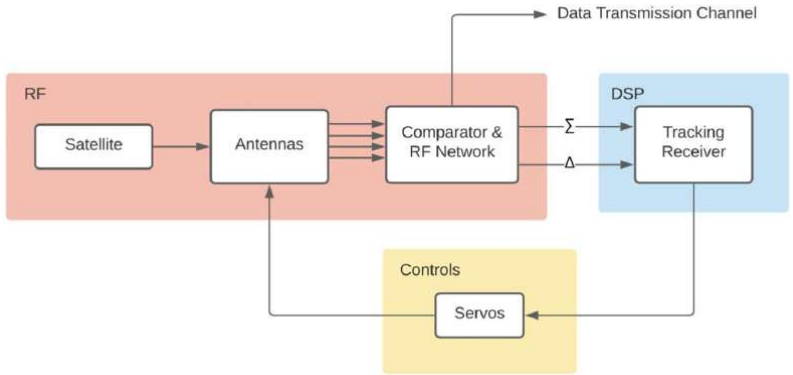
Create training data that will hold data points generated from the existing RF model

Develop method of extracting data from the RF model

Implement ML algorithm to guess updated PID parameters

Existing Infrastructure

- Matlab/Simulink model that simulates dynamic response of antenna tracking system in time domain
  - Model is used to reliably generate training data for the ML algorithm
- Work is in close conjunction with AMEND Analog team
  - Analog team can at any point provide us with an updated model to generate more accurate data



Block diagram for existing RF model

Ideation and Tradeoffs

- Solution uses MATLAB to generate a database and uses Python libraries such as TensorFlow to run an RNN algorithm
  - Implementing solution within MATLAB toolbox is not computationally ideal
  - RNN feeds output back into MATLAB input
- Got rid of D block

Solution Block Diagram

- How fast is update?
  - Approximately 1 kHz for new reference satellite positions
  - Minutes to hours for control parameters.
- Error constraint – aim to implement upper threshold on RMS error in order to systematically tune the tracking system's desired accuracy.

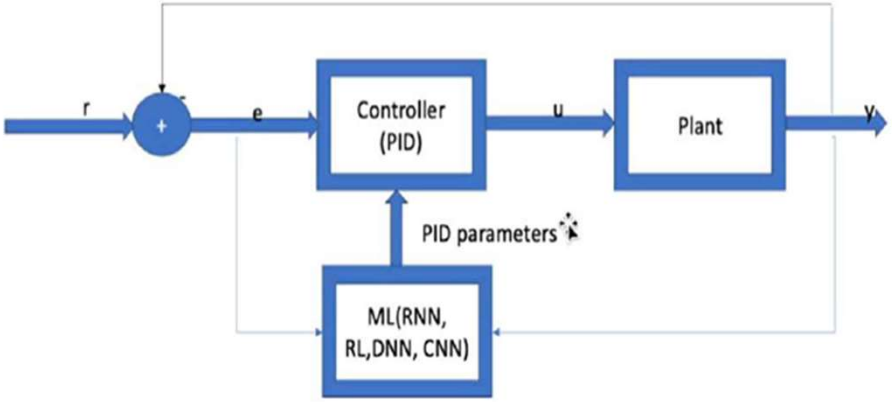
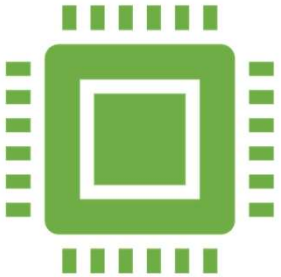


Figure 2. Block Diagram for proposed solution

Data Generation

RMS error – incorporates both signal overshoot and damping delay in its calculation – optimal for training ML algorithm to decrease both nonidealities



Classical control system design can be used to generate controller parameter datasets used to train the ML block.

Data Generation

- We apply variations to the eigenvalue "Lambda“
- There is a direct dependence of Kp and Ki on this parameter.
- We generate different errors, outputs, Kp and Ki's, and RMS errors.

```
%% Determine controller constants K1 and K2
scriptA = [Ad zeros(4,1); Tctrl*C 1];
scriptB = [Bd; 0];
K = acker(scriptA, scriptB, exp(-Lambda*Tctrl).*ones(1,5));
K1 = K(1:4); % State feedback gains.
K2 = K(5); % Error feedback gain.
```

Relationship between Kp, Ki, and Lambda

Data Generation Code

- Chooses different Lambda values to generate new K1 and K2 values
- Runs simulation with updated K1 and K2
- Writes RMS of output angle, as well as corresponding K1 & K2 to training data

```
%big loop generating training data
for Lambda = 0:10000 %stopped at 2990
    [K1, K2] = init_controller_params(Lambda);
    out = sim('slow_time'); %run simulation wi
    gen_data(out,K1,K2);    %generate data witl
end
py_code
```

```
%after a new Lambda has been set
function gen_data(out,K1,K2)
   %calculate RMS for angle_1
   sum = 0;
   angle_1_vector = out.yout{1}.Values.angle_1.Data;
   theta_test_sat = angle_1_vector(end);
   for i=1: length(angle_1_vector)
       sum = sum + (angle_1_vector(i)-theta_test_sat)^2;
   end
   RMS = sqrt(sum);

   %write R (theta_test_sat), RMS, K1, K2 to training data file
   vec_out = [theta_test_sat RMS K1 K2];
   writematrix(vec_out,'data_train.csv', 'WriteMode', 'append');
end
```

Python script generating Kp, Ki

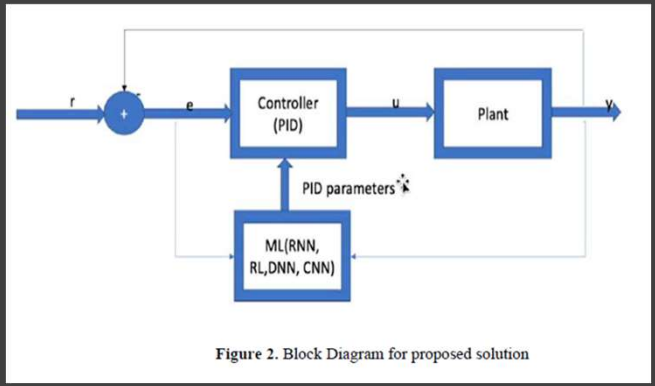


Figure 2. Block Diagram for proposed solution

- **Goal**: Take in the output of the model as input (angle_m, angle_l, wm, wl) and output Kp and Ki.
- The script will train the RNN algorithm by comparing the RMS error in the plant's output to those present in the list of training data. An upper limit can be placed on the RMS error as a constraint on the output Kp and Ki values.
- The training data comprises of angle_l RMSE, K1, K2, reference angle (r), and Lambda.
- Chooses ideal K1 and K2.

```
#we have four columns in csv (determine which column is which
dataset_train = pd.read_csv('out.csv')
training_set = dataset_train.iloc[:,1:4]
print(training_set)

sc = MinMaxScaler(feature_range = (0,1))
training_set_scaled = sc.fit_transform(training_set)

X_train = []
Y_train = []
for i in range (60,1258):
    X_train.append(training_set_scaled[i-60:i,0])
    Y_train.append(training_set_scaled[i,0])
X_train, Y_train = np.array(X_train), np.array(Y_train)
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],1))
#number 6
#first layers
regressor = Sequential()
```

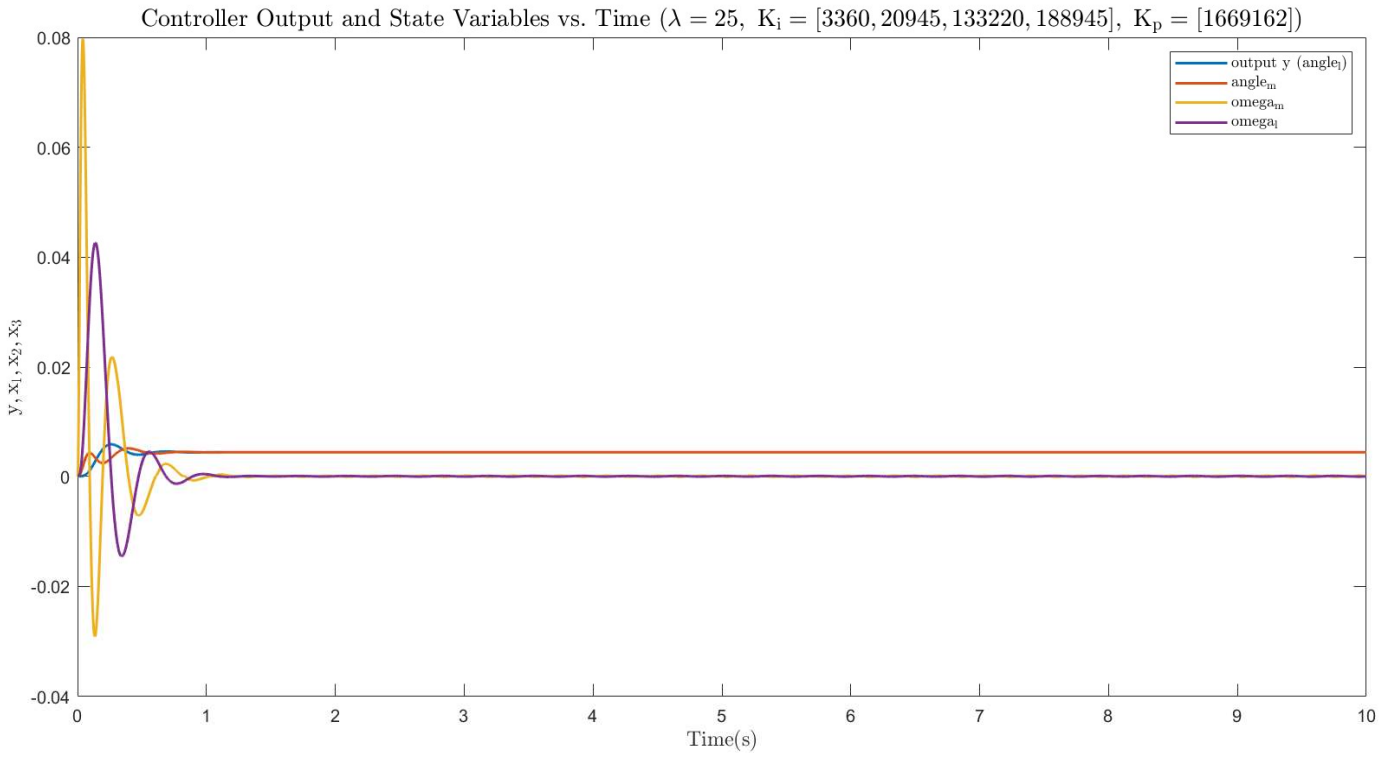
Current RNN code

- We set up RNN skeleton code that imports desired libraries
- Code uses “out.csv” as input and pulls data out to be the training set
  - 10000 step time series for angle_l, angle_m, wm, wl

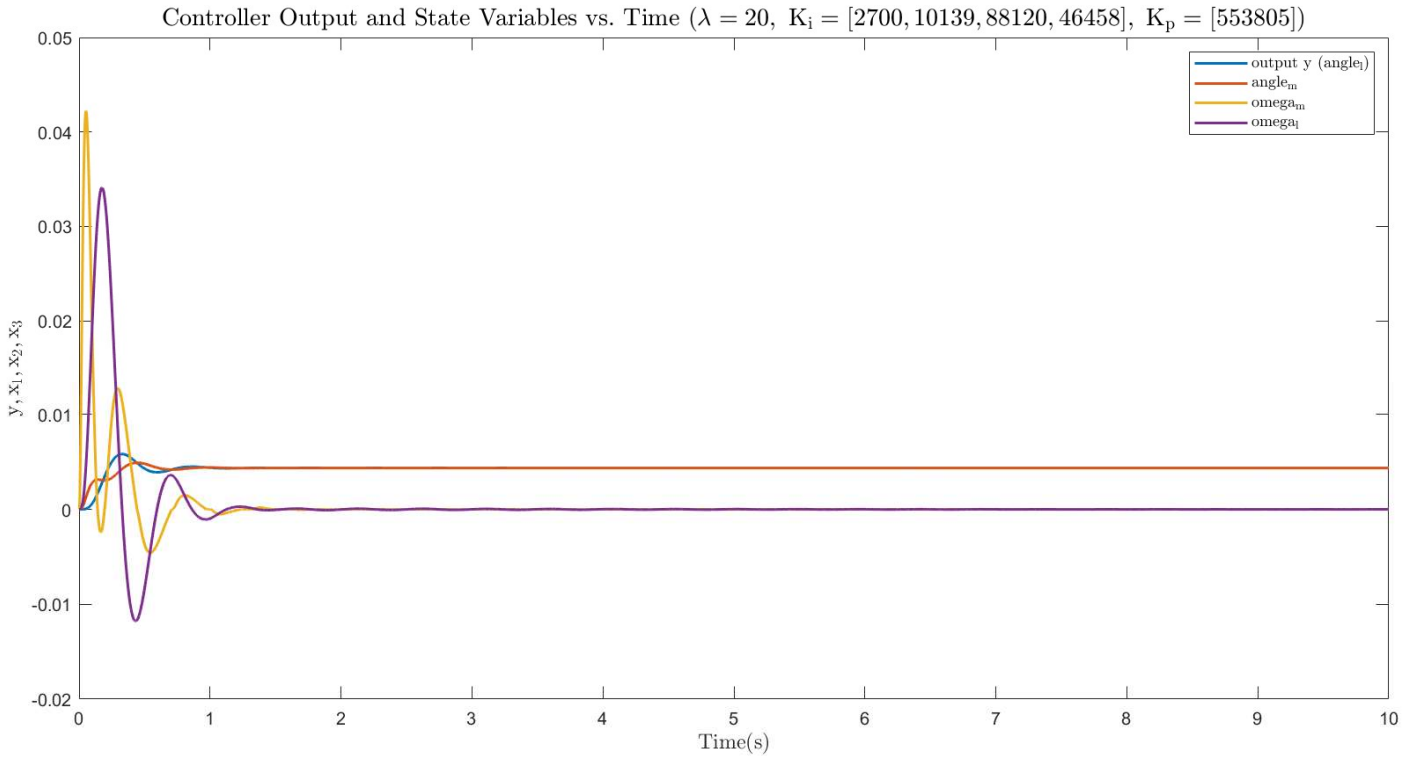
- The Machine Learning block shall be utilized to optimize this position correction process by setting a threshold RMS error as an upper limit on the accepted amount of error that can be tolerated by the control system.

- We can use the updated plant from the hardware team.

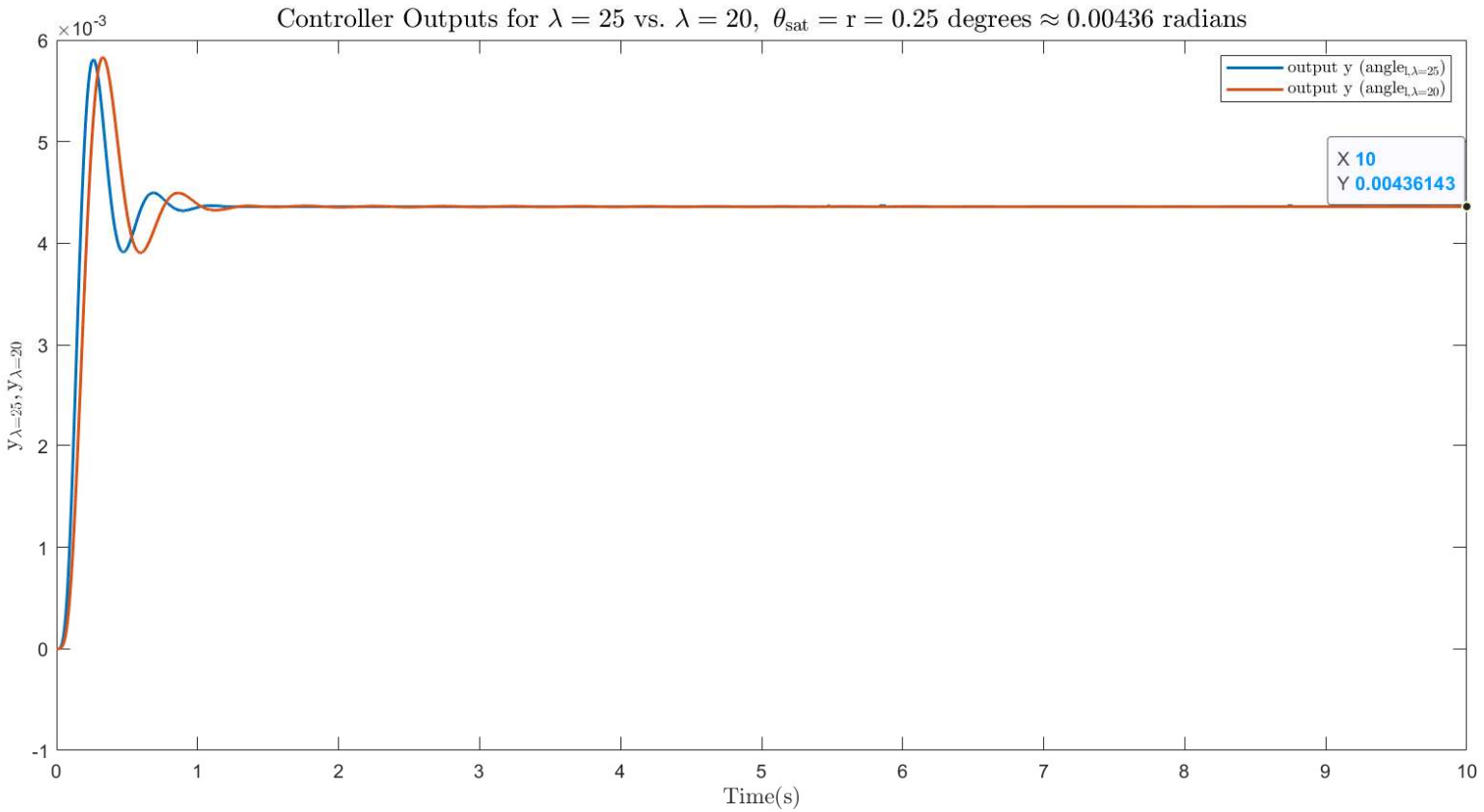
Plant Output vs Time



Plant Output vs Time



Plant Output vs Time



Questions?