VIASAT AMEND Project

Final Design Review

Machine Learning Team

4/19/2022

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.

Why is this useful?

- Minimize the amount of time between the acquisition of satellite position data and the execution of control system protocols for corrective actions to the ground station’s servo motors.
- Accounts for dynamic satellite data through automatic initialization of control parameters.

Main steps to take:

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

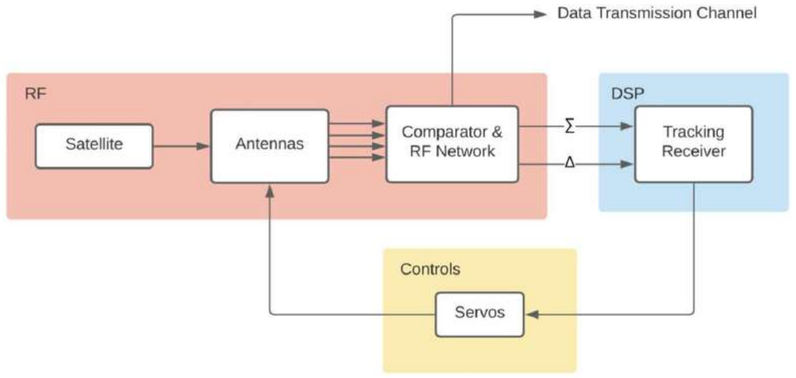
Develop method of extracting data from the RF model

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



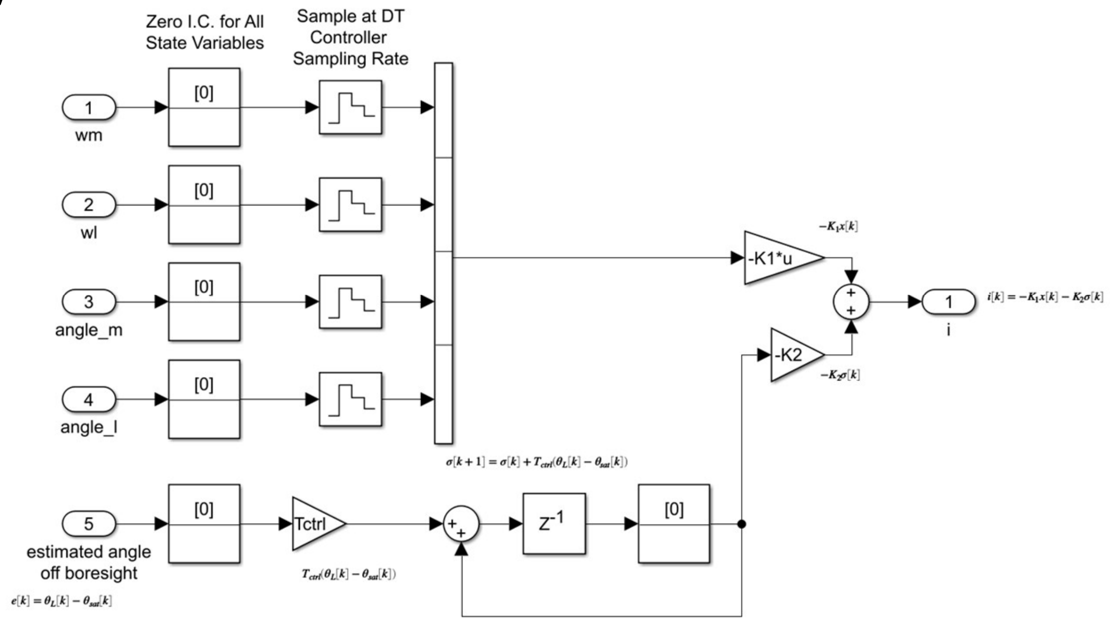
Block diagram for existing RF model

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 the Derivative (D) block, but it can be added back later

Defining plant outputs

- angle_l --- elevation angle
- angle_m --- azimuthal angle
- wl, wm --- angular frequencies in rad/s


Zero I.C. for All State Variables
Sample at DT Controller Sampling Rate
1
wm
2
wl
3
angle_m
4
angle_l
5
estimated angle off boresight
$e[k] = \theta_L[k] - \theta_{sat}[k]$
[0]
Tctrl
$T_{ctrl}(\theta_L[k] - \theta_{sat}[k])$
$\sigma[k+1] = \sigma[k] + T_{ctrl}(\theta_L[k] - \theta_{sat}[k])$
z^{-1}
-K1*u
$-K_1x[k]$
-K2
$-K_2\sigma[k]$
1
i
$i[k] = -K_1x[k] - K_2\sigma[k]$

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.

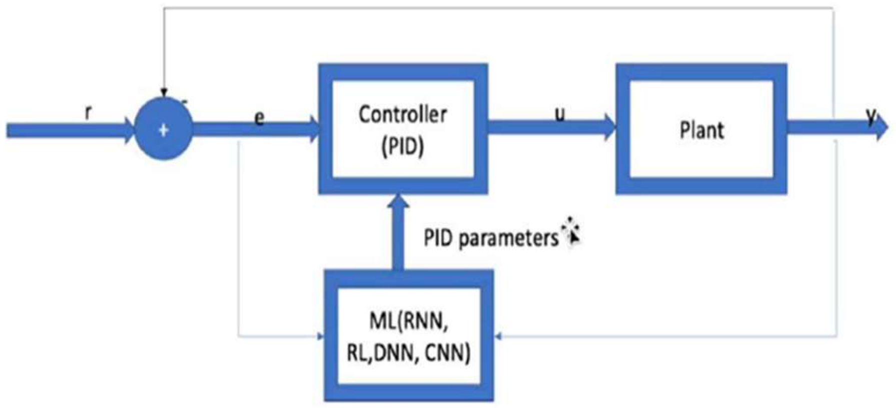


Figure 2. Block Diagram for proposed solution

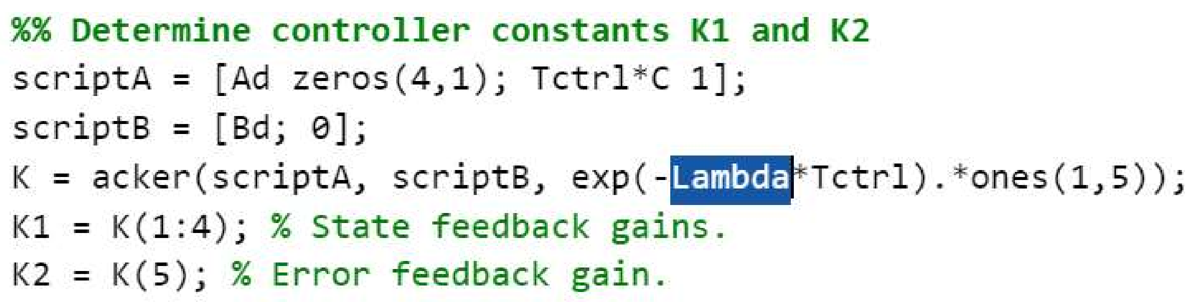
Data Generation

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.

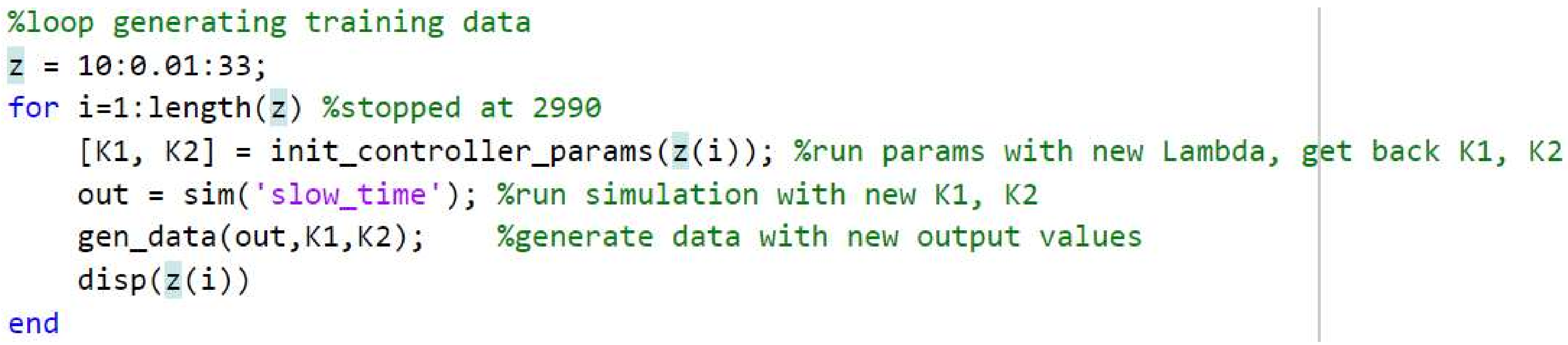


```
%% Determine controller constants K1 and K2
scriptA = [Ad zeros(4,1); Tctrl*C 1];
scriptB = [Bd; 0];
K = acker(scriptA, scriptB, exp(-Lambda*Tctrl).*ones(1,5));
K1 = K(1:4); % State feedback gains.
K2 = K(5); % Error feedback gain.
```

Relationship between Kp, Ki, and Lambda

Data Generation Code

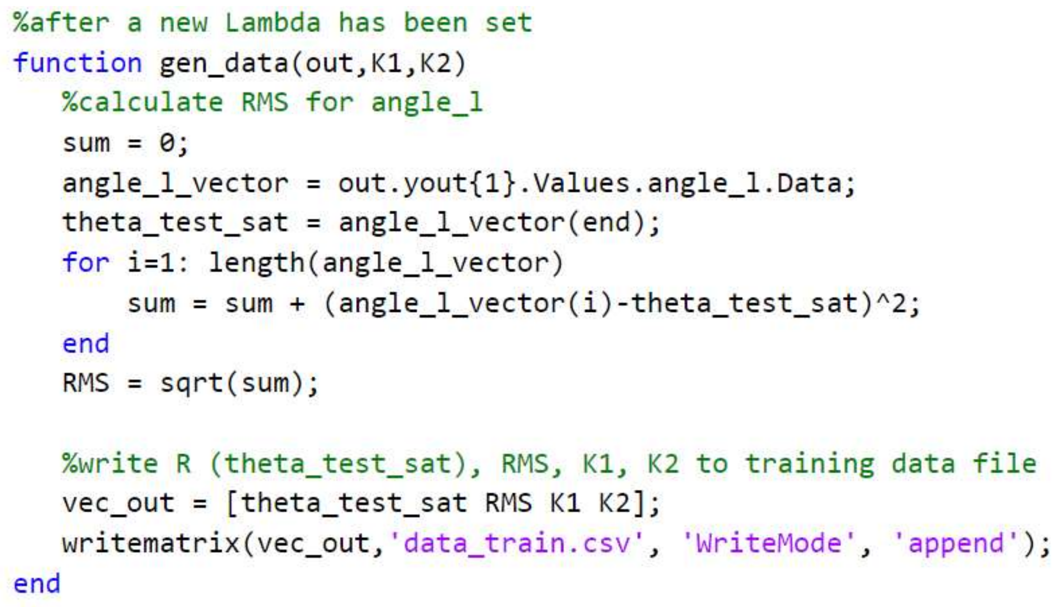
- Chooses different Lambda values to generate new K1 and K2 values
- Runs simulation with updated K1 and K2
- Writes RMS of output angle, as well as corresponding K1 & K2 to training data



```
%loop generating training data
z = 10:0.01:33;
for i=1:length(z) %stopped at 2990
    [K1, K2] = init_controller_params(z(i)); %run params with new Lambda, get back K1, K2
    out = sim('slow_time'); %run simulation with new K1, K2
    gen_data(out,K1,K2);    %generate data with new output values
    disp(z(i))
end
```

Data Generation Code

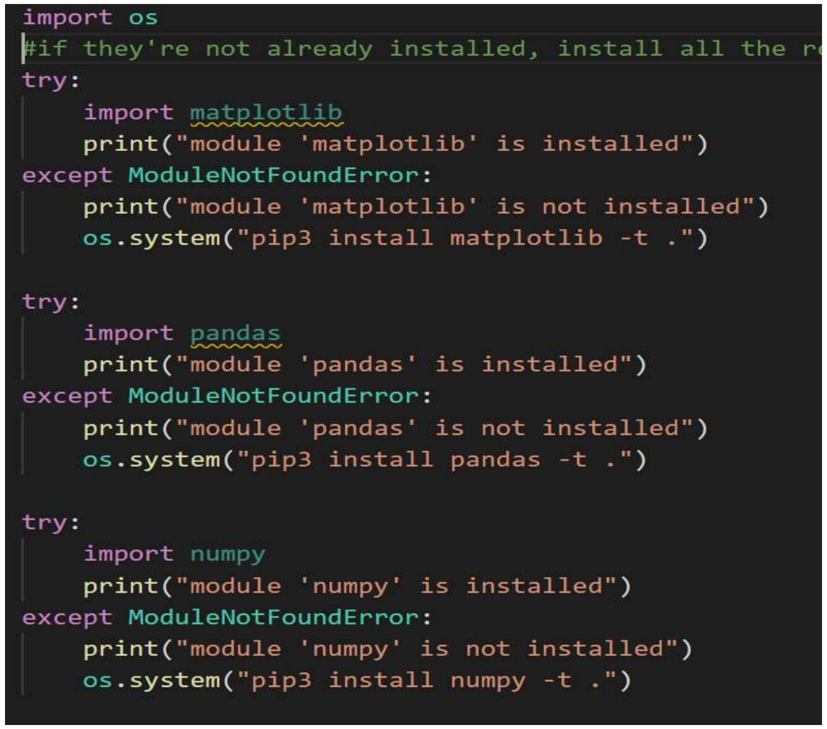
- Writes RMS of output angle, as well as corresponding K1 & K2 to training data



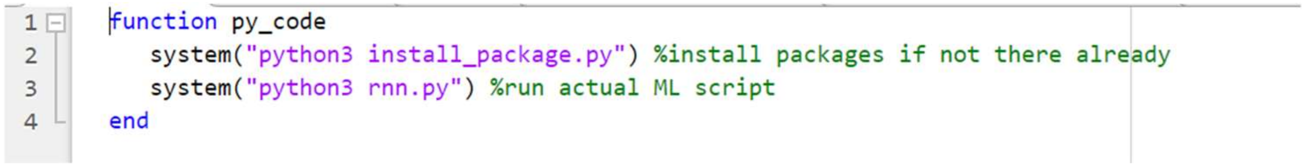
```
%after a new Lambda has been set
function gen_data(out,K1,K2)
   %calculate RMS for angle_l
   sum = 0;
   angle_l_vector = out.yout{1}.Values.angle_l.Data;
   theta_test_sat = angle_l_vector(end);
   for i=1: length(angle_l_vector)
       sum = sum + (angle_l_vector(i)-theta_test_sat)^2;
   end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

Create Python Environment

- Set up libraries such that the ML algorithm can be run directly from MATLAB
- Only requires Python to be installed on machine
- Data generation will create .csv files in location
- Python code will run in Matlab, take training data, and generate results



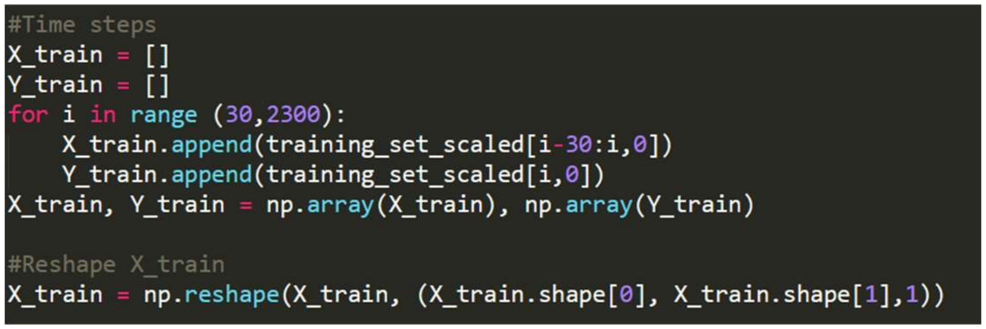
```
import os
#if they're not already installed, install all the r
try:
    import matplotlib
    print("module 'matplotlib' is installed")
except ModuleNotFoundError:
    print("module 'matplotlib' is not installed")
    os.system("pip3 install matplotlib -t .")

try:
    import pandas
    print("module 'pandas' is installed")
except ModuleNotFoundError:
    print("module 'pandas' is not installed")
    os.system("pip3 install pandas -t .")

try:
    import numpy
    print("module 'numpy' is installed")
except ModuleNotFoundError:
    print("module 'numpy' is not installed")
    os.system("pip3 install numpy -t .")
```



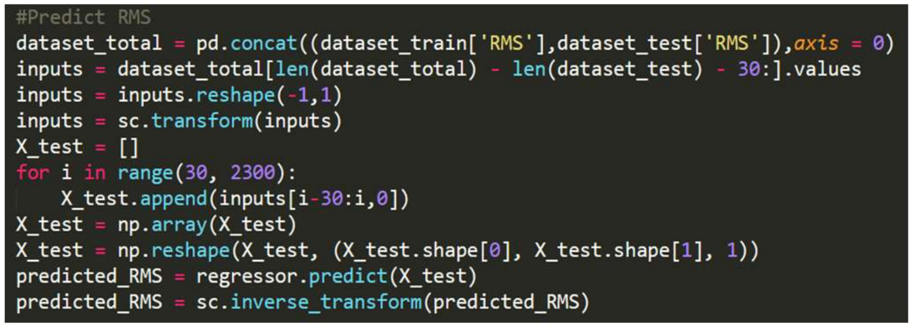
```
function py_code
   system("python3 install_package.py") %install packages if not there already
   system("python3 rnn.py") %run actual ML script
end
```

Python script generating Kp, Ki

- **Goal**: Take in the output of the model as input (angle_m, angle_l, wm, wl) and determine Kp and Ki through optimal lambda.
- The script trains 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 dataset comprises of angle_l RMSE, K1, K2, reference angle (r), and Lambda.
- Chooses ideal lambda to determine Kp and Ki.



```
#Time steps
X_train = []
Y_train = []
for i in range (30,2300):
    X_train.append(training_set_scaled[i-30:i,0])
    Y_train.append(training_set_scaled[i,0])
X_train, Y_train = np.array(X_train), np.array(Y_train)

#Reshape X_train
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1],1))
```

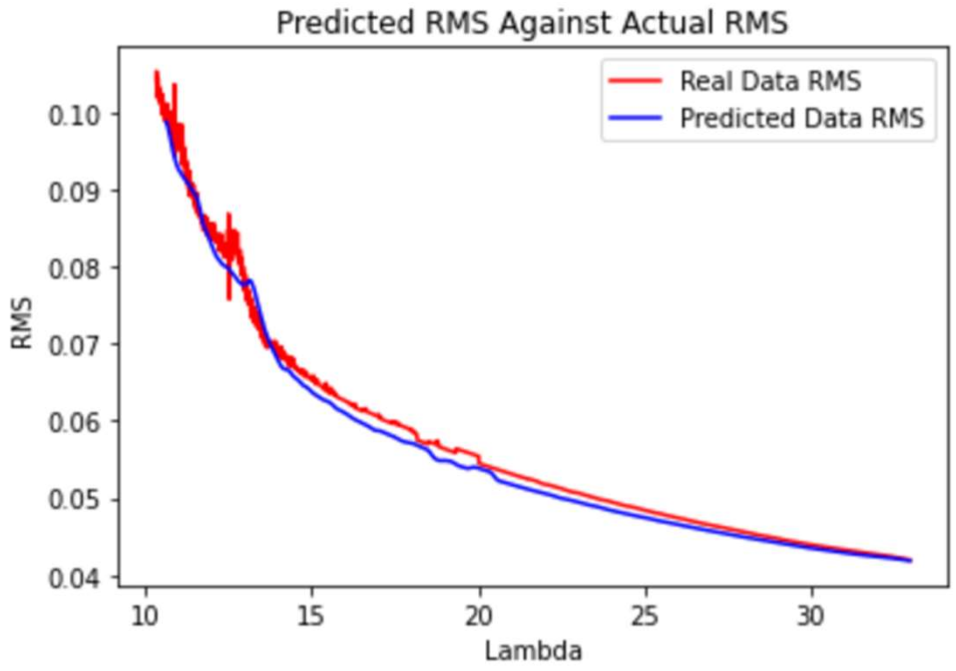


```
#Predict RMS
dataset_total = pd.concat((dataset_train['RMS'],dataset_test['RMS']),axis = 0)
inputs = dataset_total[len(dataset_total) - len(dataset_test) - 30:].values
inputs = inputs.reshape(-1,1)
inputs = sc.transform(inputs)
X_test = []
for i in range(30, 2300):
    X_test.append(inputs[i-30:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
predicted_RMS = regressor.predict(X_test)
predicted_RMS = sc.inverse_transform(predicted_RMS)
```

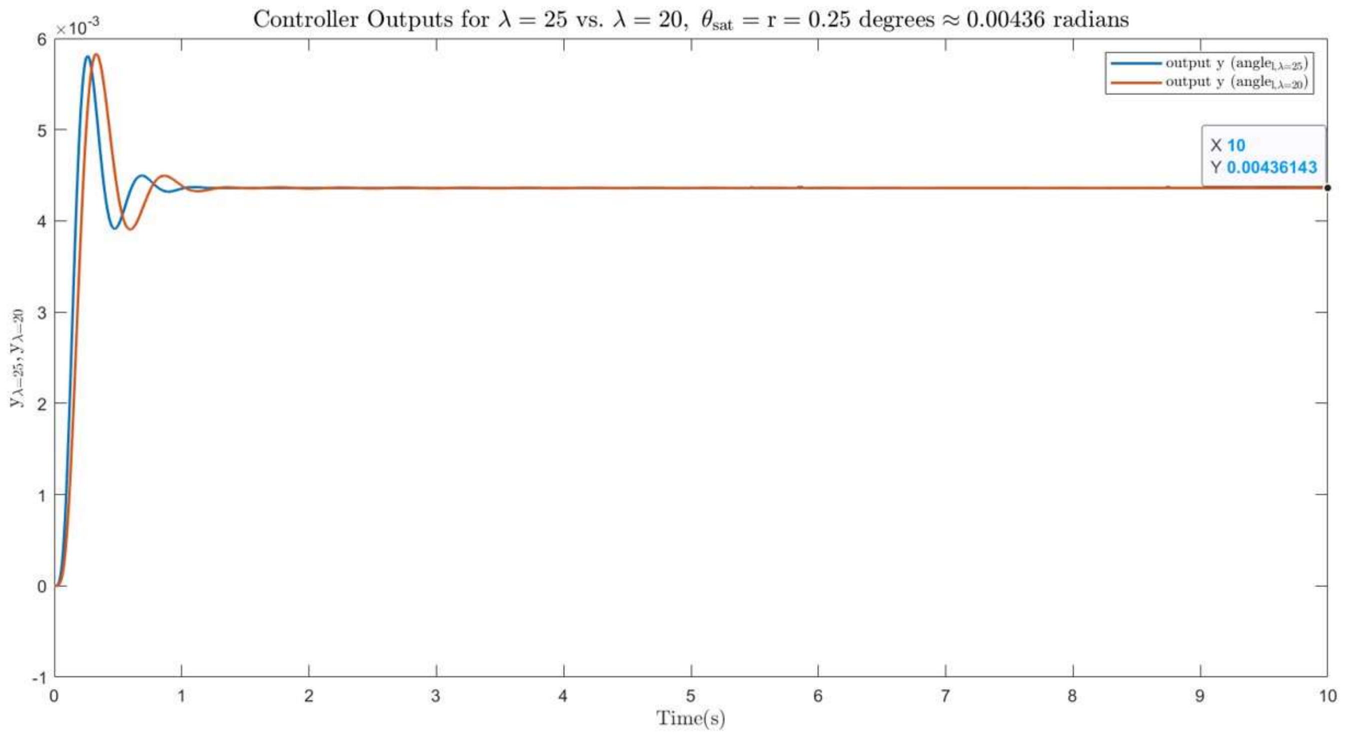
Current RNN code

- We set up RNN code that imports desired libraries
- Code uses "out.csv" as input and pulls RMS 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.
    - Output is a prediction of the RMS at a lambda value to be used for optimal Ki and Kp
- We can use the updated plant from the hardware team to generate datasets.

Working Results

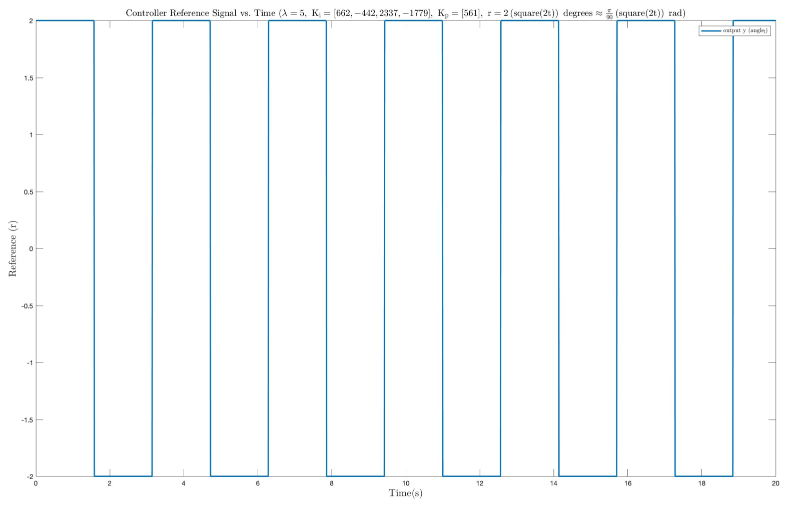


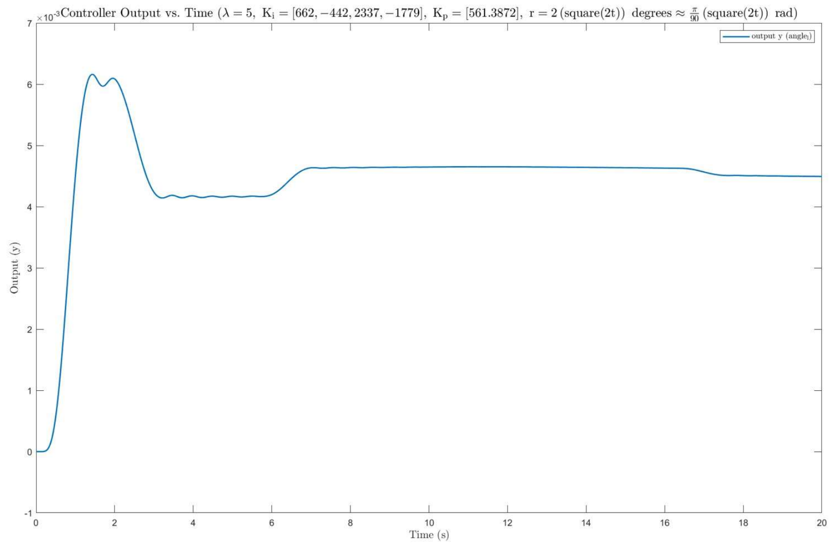
Plant Output vs Time



Adding dynamic input

- Test with functions like square and sine instead of constant reference
- Achieved some level of tracking with a square wave
- Lower lambda used -> Slower changes to track





Future Work

- Experiment with and improve control system for better dynamic tracking
- A derivative (D) block can be added in the Simulink to create a PID controller instead of the PI used in this project for higher accuracy
- A $< 30\%$ overshoot parameter can be added to the machine learning algorithm for better control

Auto-track Model with Error and Noise for Dishes (AMEND) Machine Learning Team

Undergraduate Team: Mikias Balkew, Adrija Bhattacharya, Tyler Cole, Shreyas Mhasawade, Chris Rothmann
Faculty Advisor: Dr. Xiaoli Ma

Background

- The number of Low Earth Orbit (LEO) satellites are increasing, which requires significant investment in ground stations for communications

Goal: Create a machine learning (ML) algorithm capable of generating reliable PID controller parameters for the Auto-tracking Control System designed to keep a ground station's parabolic dish pointed at a LEO satellite at all times.

Fig 1. Viasat 7m dish

Why: Minimize the amount of time between the acquisition of satellite position data and the execution of control system protocols for corrective actions to the ground station's servo motors, accounting for dynamic satellite data through automatic initialization of control parameters.

Normally controls are done with traditionally calibrated PID systems

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

Method

- The existing model from last year will produce some representative data for chosen scenarios, and the data generated will create more accurate inputs for the PID controller by using machine learning techniques for analysis

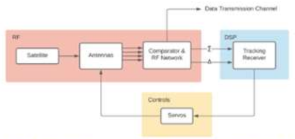


Fig 2. Block Diagram for the existing RF Model

- RMS error incorporates both signal overshoot and damping delay in its calculation and is thus optimal for ML training
- Classical control system design is used to generate control parameter data used to train the ML block

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

Develop method of extracting data from the RF model and transfer it into the ML algorithm

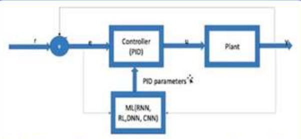


Fig 3. Block Diagram for proposed solution. The MATLAB code is used to generate data which feeds into the ML code which is run in Python

MATLAB Code

- The MATLAB script is utilized to generate different K1 and K2 values based on strategic eigenvalue placement
- Simulations are run with updated K1 & K2 parameters
- RMS of output angle and corresponding K1 & K2 are written to training data set
- RMS data serves as training data for the ML algorithm

Python Code

- The Python script trains the RNN algorithm by comparing the RMS error in the plant's output to that present in the training data
- An upper limit will be placed on the RMS error as a constraint on the output Kp and Ki values
- The Python code also graphs a predictive plot comparing the RMSE value of the training data against the test data

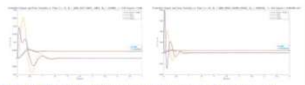


Fig 4. Graph of plant outputs as a function of time for two different lambda values

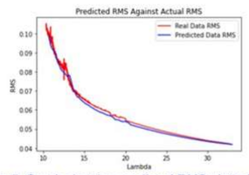


Fig 5. Graph showing predicted RMS plotted against the Lambda value to demonstrate improved RMS error through the ML algorithm

Future Work

- A derivative (D) block can be added in the Simulink to create a PID controller instead of the PI used in this project for higher accuracy
- A < 30% overshoot parameter can be added to the machine learning algorithm for better control

References

[1] Ziegler, J. G., and N. B. Nichols. "Optimum Settings for Automatic Controllers." Journal of Dynamic Systems, Measurement, and Control, vol. 115, no. 2B, 1993, pp. 220–222., https://doi.org/10.1115/1.2899060.

[2] Zulu, Andrew. "Towards Explicit PID Control Tuning Using Machine Learning." 2017 IEEE AFRICON, 2017, https://doi.org/10.1109/afrcon.2017.8095520.

[3] Davis, Zachary et al. "A Causal Model Approach to Dynamic Control." Cognitive Science (2018): 281-286.

Questions?