Lebanese University - Faculty of Engineering III

TWO-WHEELED INVERTED PENDULUM

Buraq Al Alaeli

Fouad Atwi

Presented to Prof. Hassan Shreim

Abstract

This project aims to simulate the behavior of a Two-Wheeled Inverted Pendulum (TWIP) system as well as controlling it using several approaches. These approaches being Modeled Reference Adaptive Controller (MRAC) and the Sliding Mode Controller (SMC). The system is modeled, implemented, and simulated over MATLAB/Simulink.

Computer vision algorithms are going to be used to provide an autonomous method of generating inputs to the system.

Contents

Table of Figures

CHAPTER 1: The System

The system is a two-wheeled inverted pendulum (TWIP). It consists of two motor-powered wheels connected to a base. The base consists of a mass that is offset from the wheels' axis of rotation (see **Fig 1**).

Stabilizing the TWIP system means being able to keep the base's center of mass upwards.

Controlling the system means being able to let the system move freely in an open space while keeping its stability.

Figure 1 A Two-Wheeled Inverted Pendulum

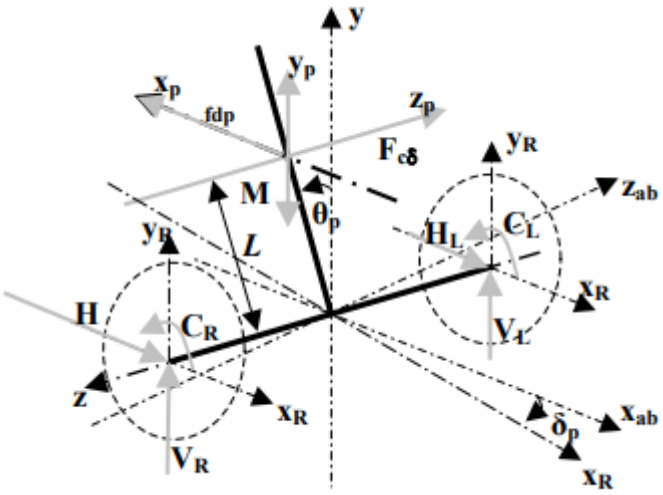

Figure 2 Free Body Diagram of chassis and wheel

The parameters used in the model are:

Symbol	Description	Value	Unit
I_p	Moment of inertia of the pendulum	0.0041	kg m^2
I_w	Moment of inertia of a wheel	0.000039	kg m^2
I_{pdel}	I_{pdel}	0.00018	kg m^2
M_p	Mass of pendulum	1.13	kg
M_w	Mass of a wheel	0.03	kg
L	Distance from axis of rotation to center of mass of pendulum	0.07	m
R	Resistance	3	Ω
r	The radius of a wheel	0.051	m
D	Distance between the two wheels	0.2	m
g	The gravity of the earth	9.81	m/s^2

Figure 3 Parameters and their values

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\theta}_p \\ \ddot{\theta}_p \\ \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \dfrac{\left(\frac{\lambda_4\Delta_1}{Z_3} - \lambda_1\right)}{Z_6} & \dfrac{\left(\frac{\lambda_4\Delta_4}{Z_3}\right)}{Z_6} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \dfrac{\left(\Delta_1 - \frac{\Delta_5\lambda_1}{Z_4}\right)}{Z_5} & \dfrac{\Delta_4}{Z_5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \theta_p \\ \dot{\theta}_p \\ \delta \\ \dot{\delta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \dfrac{\left(\lambda_3 - \frac{\lambda_4\Delta_3}{Z_3}\right)}{Z_6} & \dfrac{\left(\lambda_2 - \frac{\lambda_4\Delta_2}{Z_3}\right)}{Z_6} \\ 0 & 0 \\ \dfrac{\left(\frac{\Delta_5\lambda_3}{Z_4} - \Delta_3\right)}{Z_5} & \dfrac{\left(\frac{\Delta_5\lambda_2}{Z_4} - \Delta_2\right)}{Z_5} \\ 0 & 0 \\ -\Delta_6 & \Delta_6 \end{bmatrix} \begin{bmatrix} V_{aR} \\ V_{aL} \end{bmatrix}$$

The controllability matrix and the observability matrix of the system were calculated using MATLAB, rank(U) = rank(V) = 6 this means that the system is both controllable and observable.

Controllability matrix $U = \begin{bmatrix} B & AB & A^2B \cdots & A^{n-1}B \end{bmatrix}$

Observability matrix $V = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$

CHAPTER 2: Control Design

Model Reference Adaptive Control

Introduction

Classical controllers like PID and full-state feedback can be powerful and relatively simple in LTI systems with non-changing parameters. However, these controllers are static and do not work for nonlinear systems. One of the solutions for the problem of changing parameters and nonlinearities in the system is adaptive controllers like MRAC and MPC. The project will only focus on MRAC.

Model Reference

As the name suggests, MRAC needs a model to act as a reference to our real system. This model should be linear, stable and follows a bounded reference r.

An augmented system is created from merging our model reference system variables x with the new variable z where $\frac{dz}{dt} = y - r$.

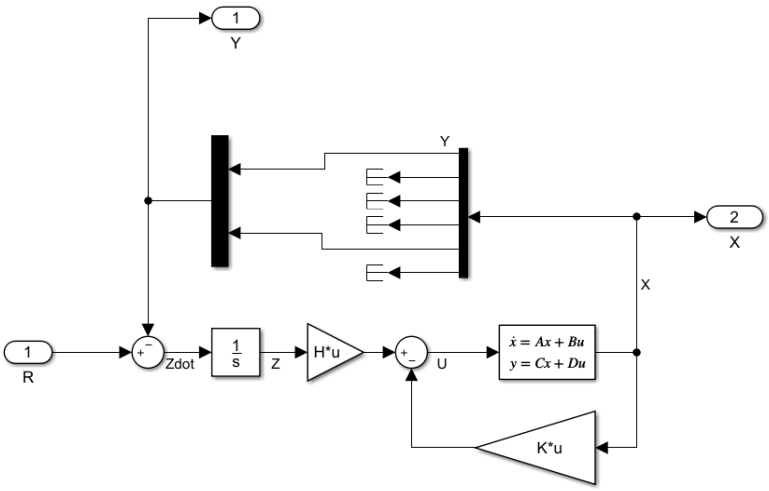

Figure 4 The augmented system

The equation of the controller is u = -Hr - Kx where H and K are calculated using the Linear Quadratic Regulator method on the augmented system.

This system represents the "perfect behavior" that the real system should follow.

Adaptive Control Implementation

The control law used in MRAC of n[th] order systems is in the form $u = \hat{k}^T{}_x x + \hat{k}^T{}_r r + \hat{\theta}^T \phi(x)$ where $\hat{k}_x \hat{k}_r \hat{\theta}$ are online estimated parameters. However, in our project $\hat{\theta}^T \phi(x)$ is omitted due to the difficulty of getting the nonlinearities.

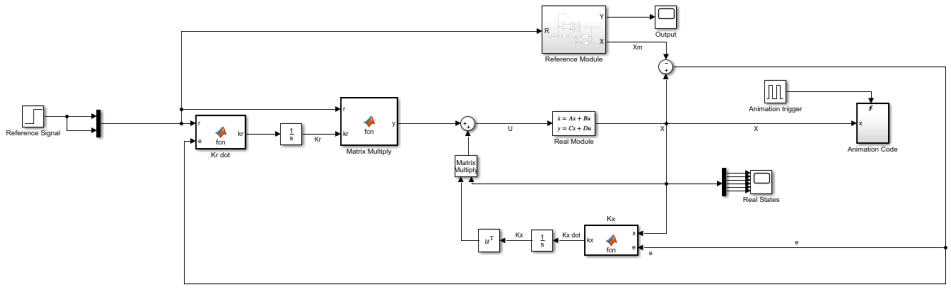

Figure 5 Model reference adaptive controller model

We estimate $\hat{k}_x$ and $\hat{k}_r$ such that the error dynamics are stable and the error between the model reference and the real reference tends to zero as time tends to infinity.

The expressions of $\hat{k}_x$and $\hat{k}_r$ are:

$$\hat{k}_x = -\Gamma_x x e^T P B sgn(\Lambda)$$

$$\hat{k}_r = -\Gamma_r r e^T P B sgn(\Lambda)$$

Where P is the solution of the Lyapunov equation and $\Lambda$is an unknown diagonal matrix.

Results

After applying the control law (as shown in **Fig 6**) and a step signal on the reference of x and δ, and having the mass matrix A slightly altered due to uncertainties.

We obtain the following results:

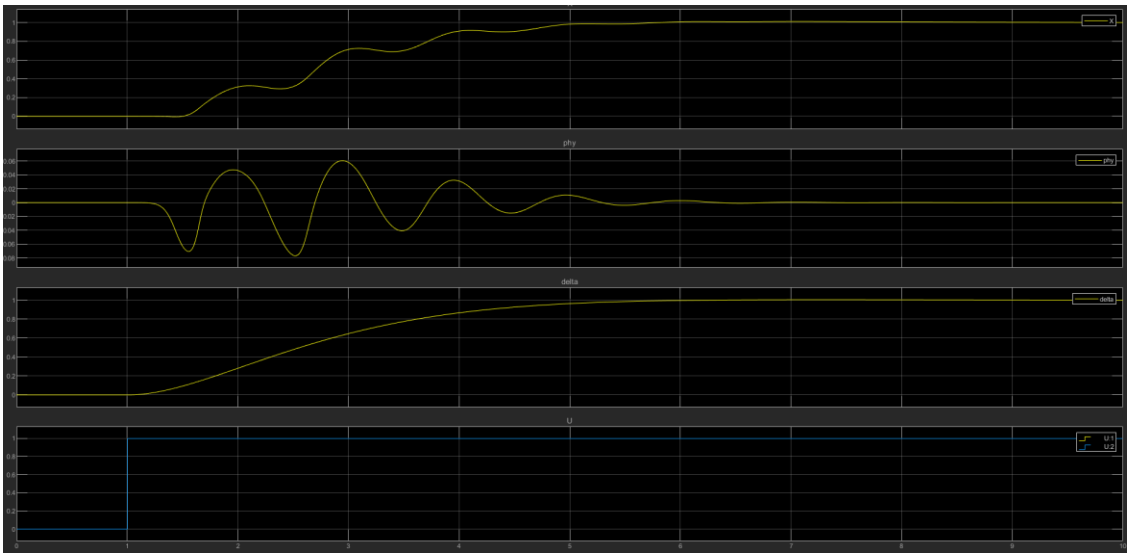

Figure 6 MRAC results

As we can see in the results, the system follows the model reference behavior and reaches the desired reference in decent timing.

Sliding Mode Controller

Introduction

As we have seen above, MRAC can be very powerful in tackling the problem of changing parameters. However, if obtaining the nonlinearities is hard or impossible, using an adaptive controller is not sufficient to get convenient results. This is when Sliding Mode Control (SMC) takes the spotlight. SMC can stabilize the system without previous knowledge of the nonlinearities, and it can negate sudden disturbances.

Sliding Mode Implementation

The sliding variable is selected as $s = B^T P x$ where P is a 4x4 positive-definite matrix.

The controller can be designed as $u(t) = u_{eq} + u_n$

$$u_{eq} = -(B^T P B)^{-1} B^T P A x(t)$$

$$u_n = -(B^T P B)^{-1}(|B^T P B|\delta_f + \varepsilon_0) sgn(s)$$

Where δ_f is upper bound of the nonlinear part f(t), and $\varepsilon_0 > 0$

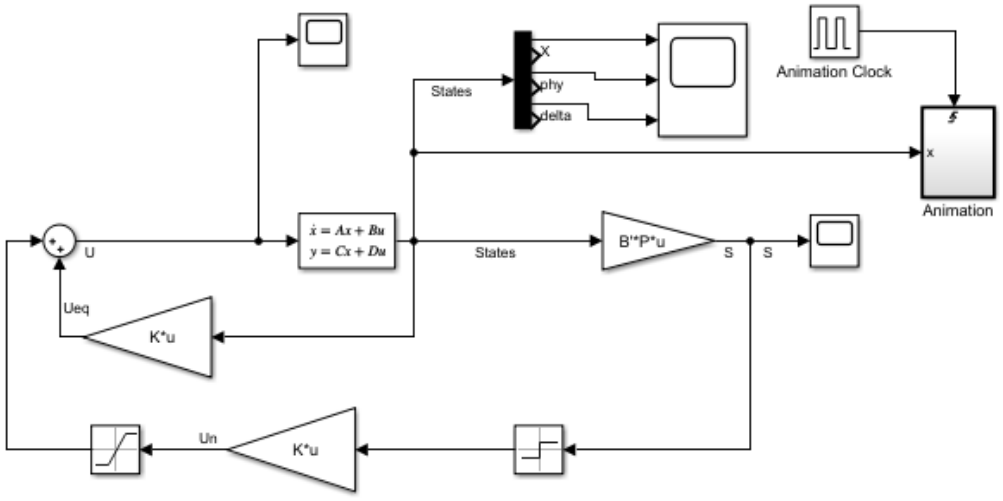

Figure 7 Sliding mode model

Results

After applying the control law (as shown in **Fig 8**) with the system's initial condition being x = 1, we obtain the results shown in the figure below:

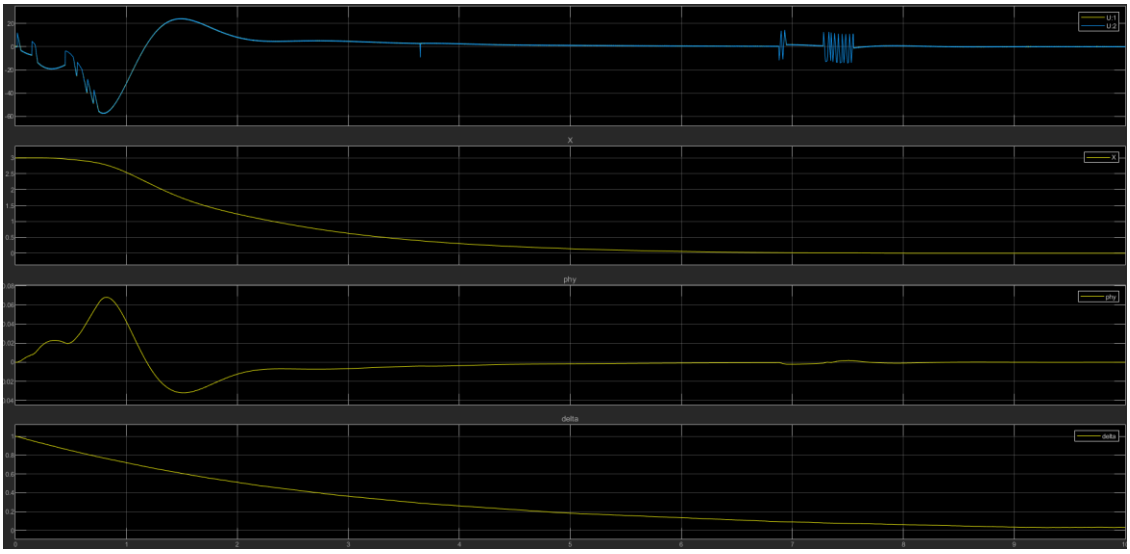

Figure 8 Sliding mode results

The sliding mode controller was successful in regulating the signal back to 0 and we notice the chattering phenomena in the input U.

CHAPTER 3: Computer Vision and Machine Lear

You Only Look Once (YOLO)

Concept

The need of one neural network to achieve object detection and classification at the same time led to the invention of YOLO (by 2016). It is a CNN that is applied one time on the image to predict all bounding boxes. To get smoother object detection results and decrease latency, one can use YOLO because it results in continuous detection with higher speed compared to RNN and fast RNN. However, it has a slightly less accuracy than faster RCNN.

YOLO is based on splitting an image into several grids, where each grid will predict the accuracy of bounding boxes representing the probability of getting object in these boxes. In addition, the grids may predict the class probability which is based on conditional probability. Then, thresholding is applied on the class probability multiplied by the accuracy of the boxes to cancel the bounding boxes that have low probabilities of containing objects of any class.

This results in training one neural network to be whole detection pipeline that is of same speed as classification pipeline. It learns things like which objects tends to co-occur together, the relative size, and the location of object.

Based on getting an image and ground truth labels for these images, then trying to match these labels with their specific grid on the image. This is achieved by determining the center of these labels and marking the cells where the center full in to work on its predicting bounding boxes. Then increase the confidence

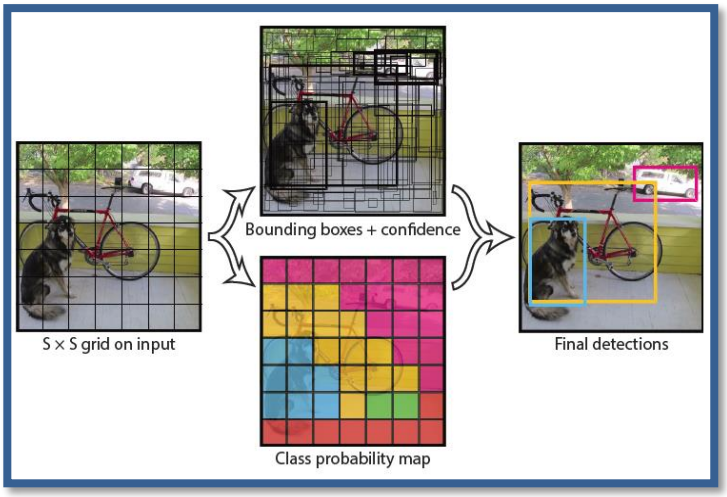

Figure 9 YOLO

of the bounding boxes that overlap with the ground truth label, as well as adjusting the probability class of that cell. All the bounding boxes of cells that do not contain any objects must be decreased in confidence.

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. The network has 24 convolutional layers followed by 2 fully connected layers, and 1 × 1 reduction layers followed by 3 × 3 convolutional layers.

YOLO was trained on the PASCAL VOC dataset that have 20 different classes like: Bicycle, Boats, Cars, Cats, Dogs, People…

The versions of YOLO: YOLO, YOLOv2 also known as YOLO 9000, YOLOv2 is faster, it understands more generalized object representations. YOLO was written by Darknet, which is an open-source neural network library written in C.

Since we are going to deal with real time object detection, we choose YOLO version 3, which will give us adequate detection in real time.

Choosing a Deep Learning Framework

YOLO is a deep learning algorithm, so it does not need any installation, what we need instead is a deep learning framework (Darknet) to run the algorithm. It is the framework built by the developer of YOLO and made specifically for YOLO.

Advantage: fast, can work with GPU or CPU
Disadvantage: only works with Linux

Data Set

Construct a data set where all the objects you want to detect are determined by a labeled bounding box. Open Images Dataset is an open source for datasets.

Implementing Pre-trained YOLO Version 3 Weights

We will focus on the detection and results upon running the YOLO v3 based on the pre-trained weights of YOLO on the COCO dataset that is composed of 80 classes.

After an image is taken, we import it into the pretrained YOLO network. The output of this network is an image that shows the bounding boxes around the detected object with its most probable class. The network also outputs the estimated distance of the obstacles which are going to be used as a reference to the control system.

Conclusion

We were able to study and control the two-wheeled inverted pendulum using two types of controllers: MRAC and SRC. Using the YOLO algorithm, we were able to generate the reference signal to the control systems in real-time.

As a further improvement, one can use different control methods like model predictive control or fuzzy logic controller.

Appendix

Training YOLO on a custom dataset on Cloud

Before setting down the steps we should define some terms:

- Losses represent the difference between the predicted values and the actual labels, so it is a quantitative description of the mistakes done by the network in predicting the outputs.
- Iterations represent the number of times a batch of data passed though the algorithm.
- Epochs is the number of times each data of training set is passing through the algorithm, one forward pass and one backward pass of all the training examples.
- Batch size is the number of data that is passing through the algorithm together.
- Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

1- Gathering Labeled Dataset

- Using the Open Image Dataset which consists of millions of images and many datasets so we can download specific labeled classes images, considering the number of images of each class as well as changing the format of labeling of the dataset to be similar to that of YOLO v3 labeling format.
- The second method is to manually label all the data after gathering it and there are many tools that helps to do that.
- Always we create a zip file concerning the real images (.jpg) we gather and the .txt files that represent the labels of each images. So, by that the storage is reduced, sharing this file become easier and become faster to be uploaded into your Cloud VM.

2- Dataset into the Cloud VM

- This is necessarily to access it and train your model on it to reduce the time of fetching files. Upload the zip file into your google drive, then make a copy from the drive to your Cloud VM and unzip it into the data folder.

3- Configuring Files for Training

- Edit the config file that is present upon downloading Darknet, after copying it into your google drive.

- Work on editing the number of batches, i.e. how many images to be trained through the model at one time according to the computing power you have.
- Change the number of classes for the YOLO layers and the number of filters of the convolutional layers depending on the number of classes, i.e. the filter of convolutional layers represents the mapping over the whole image to extract features.
- After that copy the edited custom configuration file to the cfg folder of darknet.

4- Get the obj.names and obj.data files

- Obj.names is the text files that represent each class on a line
- Obj.data represents legends of everything, i.e. where the training data and the test data and the obj.name and the number of class. As well it represents the path where the training weights will be saved.
- After that copy, these 2 files from the google drive to the data directory.

5- Get the train.txt for the data you will train your model according to it:

- There is a python file(.py) that will achieve this: any python file has to be uploaded to the google Colab notebook before running it on Colab. So, download this file and put it on the drive then copy this file to the darknet directory.
- To run any python script by a command line we can only use the command: !python path where path represent the position where this file is present in.

6- Download the pre-trained weights:

- Download the pre-trained weights of the convolutional layers of YOLOv3. Using these weights helps in making the training of your custom object detector be more accurate way faster as your model does not start from the net zero. So, download it through the command wget from the browser.

7- Train Your Costume Object Detector

- Use the darknet detector train command to train your custom object detection and that by giving obj.data, the edited configuration file and the pre-trained weights of the convolutional layers with the do not show format so it does not pop up charts showing how training is doing. Then it starts doing many iterations to get a lose under 2 and it will take a few hours to run all the training.

- You can see the lose versus iteration curve over time using the command: imshow(‘chart.png’), so you can see how your working is doing. The trained weights will be saved on our drive where we just specified in the backup of the obj.data file. So, you will not lose this waits if the Colab kick you out after 12 h or for any reason because they will be stored directly in your google drive.
- Then load the weights and use the same detector train commands for continuing from the last weights we get if you are kicked out and pick up training from where it cutoff. Keep training over and over so you can get less losses and better solutions according to the number of iterations you are taking.