

# FAST AND LOW-POWER DEEP LEARNING SYSTEM ON EMBEDDED HARDWARE FOR SELF-DRIVING AUTONOMOUS BICYCLE

A Thesis By

YUCHENG YANG

ORCID iD: 0000-0002-4077-7856

California State University, Fullerton  
Summer, 2022

---

**In partial fulfillment of the degree:**

Master of Science

**Department:**

College of Engineering and Computer Science, Computer Engineering

**Committee:**

Kiran George, Department of Computer Engineering Program, Chair  
Kenneth John Faller II, Department of Computer Engineering Program  
Pradeep Nair, Department of Computer Engineering Program

**DOI:**

10.5281/zenodo.6950722

**Keywords:**

machine learning, self-driving, FPGA, object detection, hardware optimization, low-power

**Abstract:**

Tesla, Google, and Waymo are all attempting to develop self-driving cars that can navigate real-world roadways. Many analysts anticipate that fully driverless cars will be on the road in our cities within the next five years and that practically all automobiles will be autonomous within 30 years. Automatic driving is a massive and complicated endeavor that incorporates various technology. Environment perception, behavior judgment, path planning, and motion control are the four essential automated driving technologies. Collecting and analyzing environmental and in-car data is the initial step in environmental perception, which is the foundation and premise of autonomous driving in intelligent vehicles. The optimization of image processing and the selection of hardware technologies are the key topics of this thesis. As a result, a self-driving bike is created to demonstrate the proposed software and hardware co-designed machine learning. Moreover, the proposed soft and hardware co-designed machine learning model is implemented on a development board to obtain greater energy savings and more precise data processing. To achieve this, we also propose a new implementation based on existing Xilinx software to minimize the development cost, which differs from typical FPGA transplanting technology.

## TABLE OF CONTENTS

LIST OF TABLES .....	iii
LIST OF FIGURES .....	iv
ACKNOWLEDGMENTS.....	vi
Chapter	
1. INTRODUCTION.....	1
2. BASIC PRINCIPLES AND DESIGN FRAMEWORK .....	2
Self-Driving Bicycle .....	3
Conservation of Angular Momentum.....	4
Dynamic Model .....	5
Balancing Controller Design Flow .....	6
2-Layer Filter Solution.....	7
Kalman Filter .....	7
Machine Learning on Self-driving Bicycle .....	9
3. SELF-DRIVING BICYCLE CONTROL MODULE SIMULATION.....	10
IMU Data Processing .....	10
BLDC (Brushless DC) Motor Simulation .....	11
Reaction Wheel with Stabilize Data Processing .....	13
4. IMAGE PROCESSING BASED ON MACHINE LEARNING FOR SELF-DRIVING BICYCLE...	17
Lane Processing .....	17
Object Detection.....	21
YoloV5.....	23
Tensor Processing Unit (TPU).....	25
5. HARDWARE IMPROVEMENT FPGA.....	27
Vitis IDE Instruction.....	30
VITIS A.I. ....	31
Deep-Learning Processor Unit (DPU).....	33
6. APPLICATION DEVELOPMENT AND PROCEDURES.....	34
7. APPLICATION RESULT .....	39
Resnet50 Demo .....	39
Mnsit Model Demo .....	40
8. CONCLUSION .....	41
REFERENCES .....	43

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Comparison of Deep Learning Hardware Acceleration Methods.....	41
2. FPGA & DPU Development and Software Tool Chain.....	42

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. Shows the suggested architecture for an onboard pedestrian detection system, as illustrated using a visible-spectrum camera sensor .....	1
2. Design scheme flowchart .....	2
3. Dynamic model sketch map .....	5
4. Flowchart of the combination of IMU and motor driver applies on bicycle .....	6
5. Two-Layer Filer Solution Framework.....	7
6. Self-Balancing Motor has been installed on a normal bicycle combined with Motor Driver, IMU control unit sensor, and a six cells battery; balancing function and filters are applied on an ARDUINO MEGA; Servo Motor controls the direction; D.C. motor controls run/stop status; Jason Nano + TPU compute machine learning model with collected data from the Camera .....	10
7. MATLAB script of IMU with the motor running .....	11
8. Simulation plot of IMU detection data .....	12
9. MATLAB script of IMU with the motor running .....	12
10. Simulation plot of motor running .....	13
11. MATLAB script of 2-way filter solution applies on reaction wheel .....	15
12. Simulation plots each show the time vs. Roll Angle, Roll Rate, Duty Cycle, and Reaction Wheel RPM with a 2-way Filter Solution.....	16
13. Edges of the lanes.....	18
14. Live video demo on the path .....	19
15. Training model parameter list.....	20
16. Training loss vs. Validation loss.....	20
17. Trained imaged dataset detection and prediction .....	22
18. P.R. Curves (Precision-Recall Curves) .....	23
19. The top right is the rectangles that are labeled (x,y,w,h) is the label of each instance label	
20. The plot of confusion matrix .....	24
21. TPU required the PYCPRAL library to activate .....	26
22. TPU functions required to run through the code .....	26
23. Traditional FPGA application flowchart.....	27

24.	Schematic design of traditional FPGA method from previous research .....	29
25.	Overview Platform of VITIS .....	31
26.	VITIS AI design process.....	32
27.	Basic VIVADO IP Core Design with Zynq UltraScale+ MPSoC.....	34
28.	Petalinux Configuration screenshot .....	35
29.	Petalinux generation process screenshot.....	35
30.	Flowchart of VITIS AI procedure.....	36
31.	VITIS AI main quantification functions .....	36
32.	Screenshot of VITIS AI quantification process .....	37
33.	Screenshot of VITIS AI compile process .....	37
34.	Establishment of VITIS platform (all directory paths are according to previous steps).....	37
35.	Import VITIS AI into the VITIS platform (most important step).....	38
36.	Establishment of DPU Core .....	38
37.	Resnet50 model running with DPU core (Probability reach to 98%) .....	39
38.	Mnist CNN model running with DPU core (Throughput = 3570.97fps, total frames = 2253, total time = 0.63 secs, Accuracy: 98.9 .....	40

## **ACKNOWLEDGMENTS**

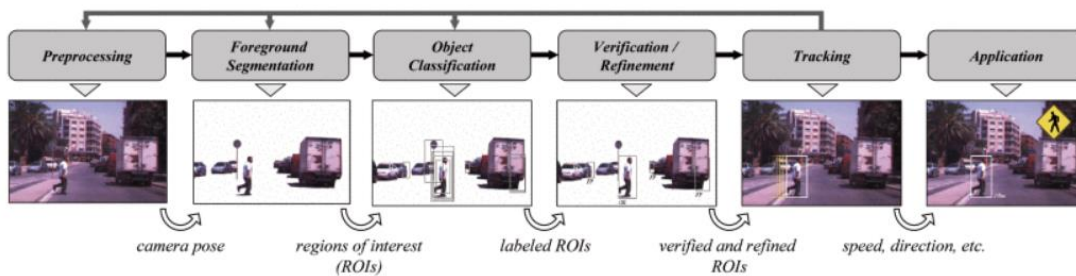
I would like to express my gratitude and appreciation to my great advisor, Dr. Yu Bai, for whom my research would not have been possible without him. I am always grateful for your patience, advice, and support during my studies. Thank you for this opportunity.

I would also like to thank Prof. Kiran George, Prof. Pradeep Nair, and Prof. Kenneth John Faller II for serving on the advisory committee and providing valuable comments and suggestions to improve my research.

## CHAPTER 1

### INTRODUCTION

With a series of autonomous system technological developments worldwide over the last decade, the race to commercialize Autonomous Cars has become more competitive than ever. Tesla, Google, and Waymo are all working on self-driving cars that can travel on public roads. Many analysts predict that fully autonomous cars will be on the road in our cities within the next five years and that by the year 2030, nearly all autos will be fully autonomous. Furthermore, Tesla has spent nearly a decade developing and enhancing its Automated Driving Systems (ADSs), which employ various advanced engineering technologies such as machine learning and computer vision. These cutting-edge technologies have considerably aided their self-driving vehicles in gaining a better grasp of the world, making the best decisions, and performing the best actions at the right moment. Many scholarly articles have been produced in the last decade because of the development of autonomous driving, and their citations are growing at an exponential rate. We can easily observe that the number of publications and citations each year has been steadily increasing since 2010, reaching a new high last year. The majority of autonomous driving overview articles, on the other hand, concentrate on a single technology topic, such as Advanced Driver Assistance Systems (ADAS), vehicle control, and visual environment perception. See Figure 1. As a result, there is a strong incentive to give readers a thorough survey of the literature on autonomous driving, covering systems and algorithms, open-source datasets, industry leaders, autonomous vehicle applications, and current issues.



*Figure 1.* Shows the suggested architecture for an onboard pedestrian detection system, as illustrated using a visible-spectrum camera sensor [31]

## CHAPTER 2

### BASIC PRINCIPLES AND DESIGN FRAMEWORK

I decided to build an autonomous driving bicycle to realize better the optimization of environmental information collection and processing in autonomous cars. Because it is impossible to transfer experimental algorithms and optimizations straight to an existing self-driving car, one advantage is the cost and time savings. Second, despite reducing four to two rounds, the core implementation approach and required hardware will not be much different, with only driving, steering, and environmental data collecting, and processing being realized. Third, it is more convenient for selecting and troubleshooting hardware. See Figure 2. After all, in regular cars, the upper application logic of the general controller VCU is more sophisticated, and the single-chip microcomputer was chosen to recreate the above fundamental tasks more efficiently. Finally, we will determine where we can enhance efficiency and reduce power consumption by changing the hardware based on the natural effect of this bicycle, and the final improvement method and experimental data will be demonstrated at the end of the thesis.

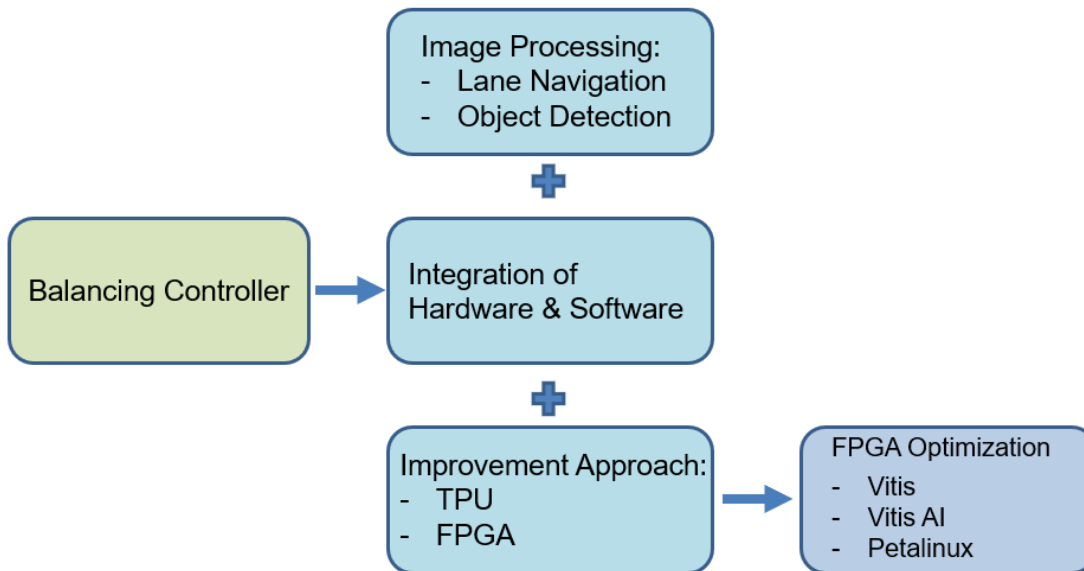


Figure 2. Design scheme flowchart



## Self-Driving Bicycle

A study on balancing a self-driving bicycle will be presented in this section. Bicycles, unlike other self-driving cars, face a particular issue when it comes to balance. Due to the little contact surface between the bike and the ground, the bicycle collapses under its own weight when standing. We can regulate the velocity of the wheel to offset the torque caused by its own weight by connecting a fast-spinning wheel controlled by a motor to the bike frame; the bike is balanced. Our algorithm, which receives the bike's roll angle, its falling rate, and the spinning velocity of the wheel to create an electric signal for the motor, controls the rotational motion of the wheel and the motor. Our simulation indicates that our algorithm can return the bike to its vertical balancing position from a tilting inclination of 1.5 degrees. After that, the bike with the wheel and the motor was put together for more testing. In actuality, the vibration from the motor interacts with the sensor, causing significant data errors. To reduce the error produced by motor vibration, we utilize a two-layer filter. The experiment data reveals that the wheel can return the bike to its vertical position. However, our method is still not stable enough to retain the bike in its balanced posture owing to inappropriate controller coefficients. As a result, a new simulation will be run with more precise specifications in order to arrive at a set of more suited coefficients.

An unstable and nonlinear system is a self-balancing bicycle robot based on the concept of an inverted pendulum. The following three main components are required to stabilize the system in this work: (1) a controller that controls the motion, which is the dynamic model of a reaction wheel (2) an IMU sensor that detects the tilt angle of the bicycle, and (3) a reaction wheel that produces reactionary torque to balance the bicycle robot. I used reaction wheel control in my design, which is when the speed of a reaction wheel is increased or decreased in order to generate a reactive torque about the spin axis, which is parallel to the bicycle frame. When the bicycle begins to fall to one side, a motor on the reaction wheel provides a reactive force on the bicycle, which restores the bicycle's balance. This system's merits include its low cost, simplicity, and absence of ground reaction, while

its disadvantages include its increased energy consumption and inability to generate large levels of torque.

### **Conservation of Angular Momentum**

One of the universal laws of physics. For example, a particle moving in a central force field is always affected by force through the center of force. Since the moment between the force and the center of force is zero, according to the angular momentum theorem, the angular momentum of the particle to the center of force is conserved [1]. Thus, the trajectory of a particle is a plane curve, and the sagittal diameter of the particle to the center of force sweeps over an equal area in equal time. If the sun is regarded as the center of force, and the planet as a particle, the above conclusion is Kepler's second law, one of Kepler's three laws of planetary motion [2]. In a system of particles not affected by external forces or external fields, the internal forces interacting with each other obey Newton's third law, so the principal moment of internal forces at any point is zero, and the conservation of angular momentum of the system is derived [3]. If the algebraic sum of the moment of a fixed axis of a system subjected to an external force is zero, then the angular momentum of the system with respect to that axis is conserved [4]. Conservation of angular momentum is also an important basic law in microphysics. In the process of decay, collision, and transformation of elementary particles, the conservation law, which reflects the universal law of nature, also includes the conservation law of angular momentum. In 1931, W. Pauli speculated that antineutrinos were produced when free neutrons decayed according to the conservation law, which was confirmed by an experiment in 1956 [5]-[7].

## Dynamic Model

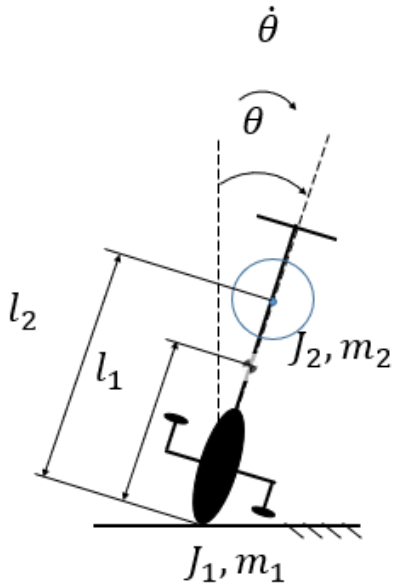


Figure 3. Dynamic model sketch map [1]

Notations:

$J_1$ - Mass moment of Inertia of the Bike about its CoG [ $kg \cdot m^2$ ]

$J_2$ - Mass moment of Inertia of the wheel about its center [ $kg \cdot m^2$ ]

$m_1, m_2$  - Mass of the bike and the wheel [ $kg$ ]

$l_1$  - Height of the bike's CoG [ $m$ ]

$l_2$ - Height of the wheel's center [ $m$ ]

$\tau_m$  - Torque generates by the driving motor [ $Nm$ ]

$\omega$  – Angular velocity of the wheel [ $rad/s$ ]

Applying Lagrange Equations for the bike and the wheel, see Figure 3, we have the following system of the equation [7], [8], [10], [14]:

$$\begin{bmatrix} m_1 l_1^2 + m_2 l_2^2 + J_1 + J_2 & J_2 \\ J_2 & J_2 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} -(m_1 l_1 + m_2 l_2)g & 0 \\ 0 & b_m \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \omega \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tau_m \quad (1)$$

However, the torque from the motor is calculated by

$$\tau_m = \frac{K_T}{R} V_m - \frac{K_T K_b}{R} \omega \quad (2)$$

Where:

- $K_T$  - Torque constant [ $Nm/A$ ]
- $K_b$  - Back EMF Const [ $\frac{V}{\frac{rad}{s}}$ ]
- $R$  – Motor resistance [ $\Omega$ ]
- $V_m$ - Supplied Voltage [ $Volts$ ]

Thus:

$$\begin{bmatrix} m_1 l_1^2 + m_2 l_2^2 + J_1 + J_2 & J_2 \\ J_2 & J_2 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} -(m_1 l_1 + m_2 l_2)g & 0 \\ 0 & b_m + \frac{K_T K_b}{R} \end{bmatrix} \begin{bmatrix} \dot{\theta} \\ \omega \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{K_T}{R} \end{bmatrix} V_m \quad (3)$$

Or matrix form:

$$\mathbf{M} \dot{\mathbf{v}} + \mathbf{V} \mathbf{v} = \mathbf{S} V_m \quad (4)$$

### Balancing Controller Design Flow

From the mathematical equation, we introduce these state-space variables, see Figure 4:

- $x_1 = \theta$
- $x_2 = \dot{x}_1 = \dot{\theta}$
- $x_3 = \omega$

We have the following state-space equation:

$$\dot{\mathbf{X}} = \mathbf{A} \mathbf{X} + \mathbf{B} u \quad (5)$$

Where:

$$\dot{\mathbf{X}} = [\dot{x}_1 \quad \dot{x}_2 \quad \dot{x}_3]^T, \quad \mathbf{A} = -\mathbf{M}^{-1} \mathbf{V}, \quad \mathbf{B} = \mathbf{M}^{-1} \mathbf{S} \quad (6)$$

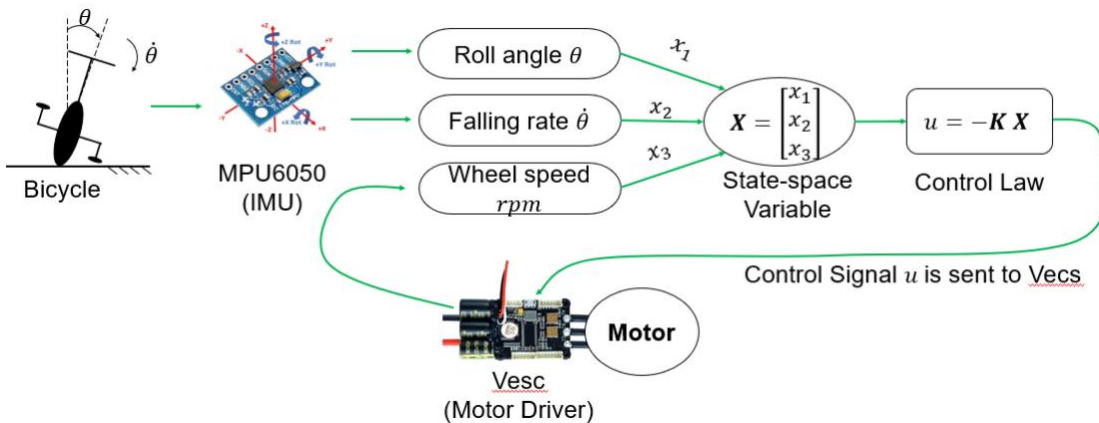


Figure 4. Flowchart of a combination of IMU and motor driver apply on the bicycle

Control law:  $u = -\mathbf{K} \mathbf{X}$

Where  $\mathbf{K}$  is designed based on Linear Quadratic Regulation (LQR) Control, which means that  $\mathbf{K}$  is chosen to minimize the following performance index [8][9][14]:

$$J = \int_0^{\infty} (\mathbf{X}^T \mathbf{Q} \mathbf{X} + u R u) dt \quad (7)$$

## 2-Layer Filter Solution

During the trial process, we applied the controller motor onto the bicycle. It indicates strong vibration when it has been activated to control the balance. After research, I figure out it's because of the “noise” element. Therefore, I added a 2 – layer filter which adds a Kalman filter in the function of controlling the balance motor, see Figure 5, which can erase the vibration [13], [14].

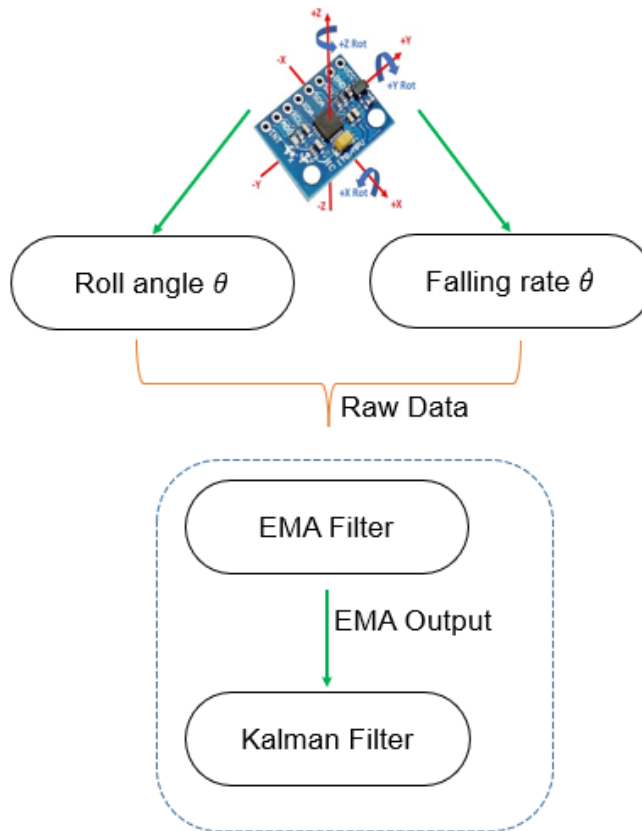


Figure 5. Two-Layer Filer Solution Framework

## Kalman Filter

The Kalman filter is a technique for estimating variables in a variety of processes. A Kalman filter is a mathematical name for a filter that estimates the states of a linear system [10]-[12]. The

Kalman filter is not only practical but is also intellectually appealing since it can be proved that it is the filter that minimizes the variance of the estimation error among all feasible filters. Kalman filters are commonly used in embedded control systems because controlling a process necessitates a precise estimation of the process variables [12]. The Kalman filter uses feedback control to estimate a process: it guesses the process state at a given time and then receives input in the form of (noisy) measurements. As a result, the Kalman filter equations are divided into two categories: time update equations and measurement update equations. The time derivative equations are in charge of projecting the present state and error covariance estimates ahead (in time) in order to produce a priori estimates for the next time step [12]-[14]. The feedback—that is, adding a new measurement into the a priori estimate to generate an improved a posteriori estimate—is handled by the measurement update equations. Kalman filter consists of three equations,

$$\mathbf{K}_k = \mathbf{A}\mathbf{P}_k\mathbf{C}^T(\mathbf{C}\mathbf{P}_k\mathbf{C}^T + \mathbf{S}_z)^{-1} \quad (8)$$

$$\hat{\mathbf{X}}_{k+1} = (\mathbf{A}\hat{\mathbf{X}}_k + \mathbf{B}\mathbf{u}_k) + \mathbf{K}_k(\mathbf{y}_{k+1} - \mathbf{C}\hat{\mathbf{X}}_k) \quad (9)$$

$$\mathbf{P}_{k+1} = \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{S}_w - \mathbf{A}\mathbf{P}_k\mathbf{C}^T\mathbf{S}_z^{-1}\mathbf{C}\mathbf{P}_k\mathbf{A}^T \quad (10)$$

each one necessitates the use of a matrix. A  $-1$  superscript in the above equations denotes matrix inversion, whereas a T superscript denotes matrix transposition. The Kalman gain is the K matrix, while the estimation error covariance is the P matrix.

The state estimation ( $\hat{\mathbf{X}}$ ) formula is straightforward. The first element is simply A times the state estimate at time k plus B times the known input at time k to calculate the state estimate at time k + 1 [9][11]. If we did not have a measurement, this would be the state guess. In other words, much like the state vector in the system model, the state estimate would propagate across time. The correction term is the second component in the ( $\hat{\mathbf{X}}$ ) equation, and it specifies the amount by which the propagating state estimate should be corrected owing to our measurement. If the measurement noise is considerable,  $\mathbf{S}_z$  will be large, thus K will be little, and we won't give the measurement y much credence when computing the following  $\hat{\mathbf{X}}$ , according to the K equation [14]. On the other hand, if the

measurement noise is low,  $S_z$  will be low; therefore  $K$  will be high, and we will give the measurement a lot of credence when computing the next  $\hat{X}$ .

### **Machine Learning on Self-driving Bicycle**

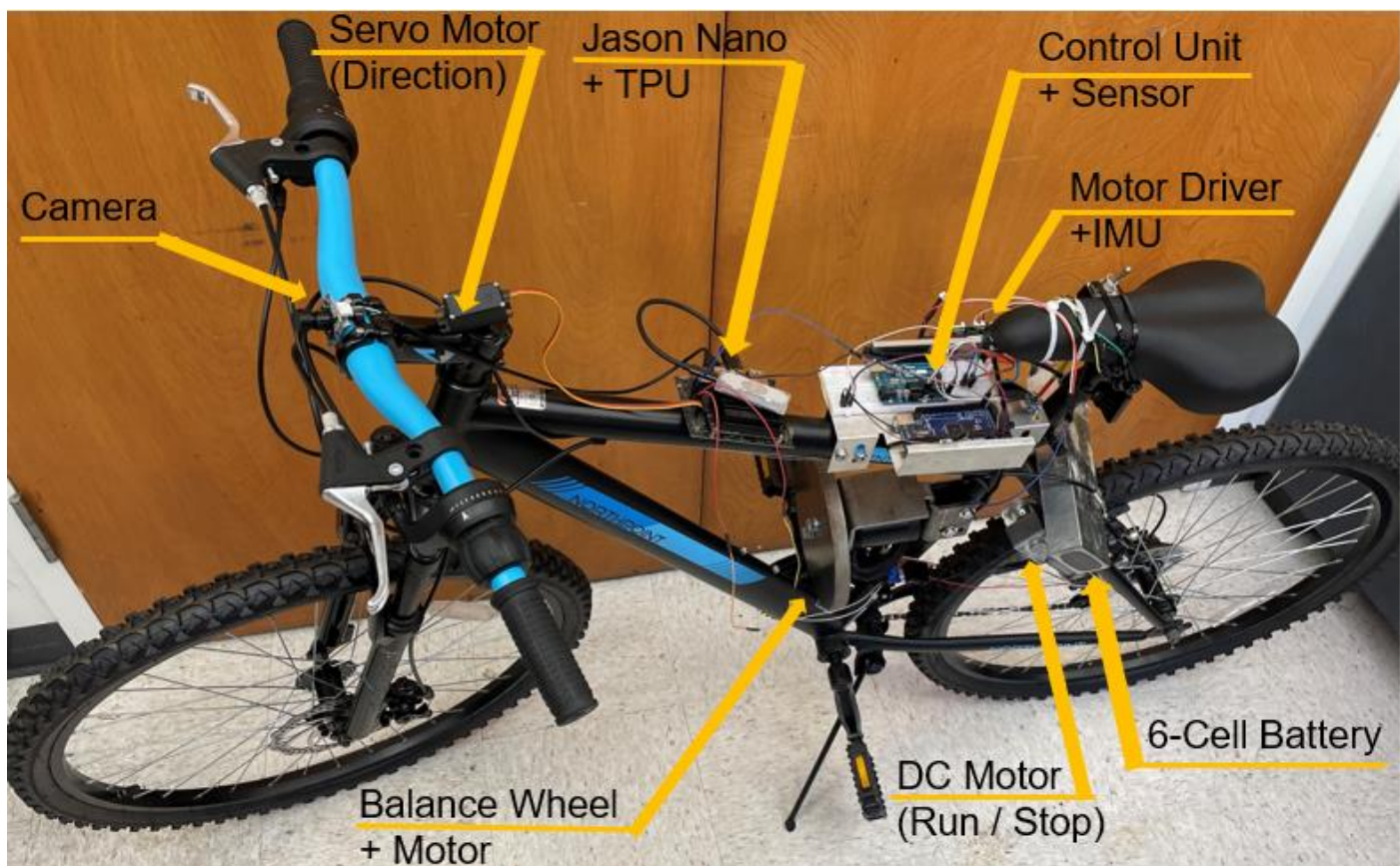
Use OpenCV to detect color, edges and lines segments. Then compute steering angles, so that bicycle can navigate itself within a lane. Lane detection's job is to turn a video of the road into the coordinates of the detected lanes. Once the line segments are classified into two groups, I just take the average of the slopes and intercepts of the line segments to get the middle line. Once I have the coordinates of the lanes, the handle of the bicycle will be steered by the control signal based on the angle of the middle line. NVIDIA Jetson nano is a convincing GPU that process efficient real-time images. A webcam is a sensor for image data collection. Servo motor controls the handle, which receives the steering signal from the lane navigation function to keep the handle straight, turn left, or turn right. The object detection function will process the machine learning model, which compares with the real-time sensor information and sends a signal to the D.C. motor to control the back wheel for running forward, backward, and stops. I set human and stop signs in my algorithm as a stop signal, which enables bicycles to avoid a collision.

## CHAPTER 3

### SELF-DRIVING BICYCLE CONTROL MODULE SIMULATION

To test the proposed control module for the self-driving autonomous bicycle, we perform careful experiments using MATLAB software, see Figure 6. The experiments are designed for several testing aspects: 1) IMU motor design; 2) Brushless DC design; 3) Reaction Wheel with Stabilize Data Processing.

The simulation clearly depicts operational data when the self-balance motor activates and how the two-way filter technology helps to steady the motor and eliminate the "noise."



*Figure 6.* Self Balancing Motor has been installed on a normal bicycle combined with Motor Driver, IMU control unit sensor, and a six cells battery; balancing function and filters are applied on an ARDUINO MEGA; Servo Motor controls the direction; D.C. motor controls run/stop status; Jason Nano + TPU compute machine learning model with collected data from the Camera.

### IMU Data Processing

Accelerometers are used to detect and report particular forces, while gyroscopes are used to monitor and report angular rates. These elements combine to form an inertial measurement unit



(IMU). See Figure 7, MATLAB can gather time at stationary/motor runs, and roll angle at stationary/motor runs using IMU and save the data as an excel file.

```
clc, clf, clear all, close all

% Extract the data from data files
% t_s - Time marks when testing MPU6050 at stationary, sec
% roll_s - Roll angle measured by MPU6050 at stationary, deg
% t_m - Time marks when testing MPU6050 when motor runs, sec
% roll_m - Roll angle measured by MPU6050 when motor runs, deg
[t_s, roll_s] = MPU6050_Stationary("MPU6050_Raw_Stationary.xlsx");
[t_m, roll_m] = MPU6050_Motor("MPU6050_Raw_withMotor.xlsx");

% Plot the raw data for both cases
figure; plot(t_s, roll_s, 'b', 'linewidth', 2);
hold on
plot(t_m, roll_m, 'g-.', 'linewidth', 1);
hold on
plot(t_s, 3*ones([length(roll_s) 1]), 'k-.', 'linewidth', 1.5)
xlabel('Time [sec]'), ylabel('Roll Angle [deg]'), title('Raw Data of Roll Angle')
legend("Without motor running", "Motor runs at 8000RPM", "True Position"), grid
```

Figure 7. MATLAB script of IMU with the motor running

See Figure 8, we can see that when the motor is running at 8000 RPM, the roll angle is moving and changing over time, indicating that the rotating motor is operating correctly and causing the bicycle to return to its normal position. However, even when the balancing motor is not operating, the blue line can still be seen to be in motion, demonstrating that the IMUs experience vibration because of their operation. Therefore, we have to resolve the issue and stop the noise. I conducted some mechanical investigation and found a solution called the Kalman Filter to address the issue.

### BLDC (Brushless DC) Motor Simulation

After setting up the mass moment of inertia of the rotor and friction coefficient of the motor, we can simulate the torque and speed for the BLDC based on the precise specs such as motor resistance, Brushless DC velocity constant, torque constant, and back emf constant (from the motor datasheet), see Figure 9. Finding issues and conducting experiments with various types of data with the aid of simulation training is quite helpful in determining the essential components of the balancing wheel.

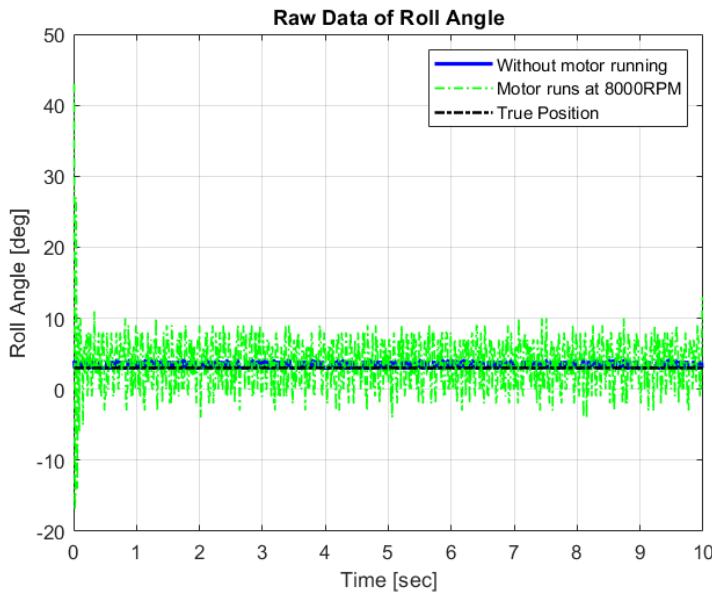


Figure 8. Simulation plot of IMU detection data

```

clc, clear all, close all
% Time for the simulation
dt = 0.001; % Sampling time, sec
t = 0:dt:0.5; % Time for the simulation, sec

% Motor electric specs
Rm = 0.5; % Motor resistance, Ohm
KV = 140; % BLDC Velocity Constant, RPM/Volt
Kt = (3/2)*(1/sqrt(3))*(30/pi)*(1/KV); % Torque Constant, Nm/A
Kb = Kt; % Back EMF Constant, V/Rad/s

% Motor mechanical spec
J_rot = 7.6624e-4; % Mass moment of inertia of rotor, kg.m^2
bm = 0.0001; % Friction coeff. of motor, N m/s /rad

% Initial conditions
w(1) = 0; % Initial motor speed, RPM

% Simulation program
Vm = 22*ones([1, length(t)]); % Supplied Voltage for the motor, Volt
for i = 2:length(t)
    w(i) = w(i-1) + dt*((1/J_rot)*((Kt/Rm)*Vm(i) - (Kt*Kb/Rm)*w(i-1) - bm*w(i-1)));
end

% Calculate the torque
Tm = Kt*((Vm - Kb*w)/Rm);

% Plot the results
figure;
subplot(2, 1, 1); plot(t, w*30/pi, 'k-', 'linewidth', 1.5);
xlabel('Time (sec)'), ylabel('Motor Speed (RPM)'), grid;

subplot(2, 1, 2); plot(t, Tm, 'r--', 'linewidth', 1.5);
xlabel('Time (sec)'), ylabel('Torque (Nm)'), grid;

```

Figure 9. MATLAB script of IMU with the motor running

See Figure 10, it demonstrates that with motor speed increases, torque production decreases.

Therefore, the balance wheel needs additional torque at the start of the launch. Additionally, as the system runs longer, less torque will be needed, which will result in lower power usage.

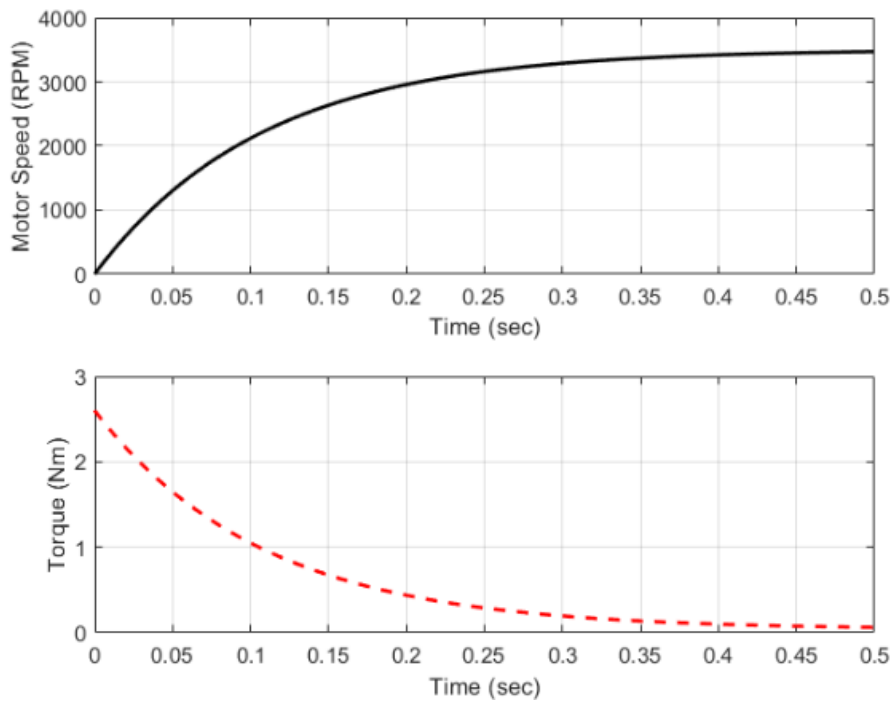


Figure 10. Simulation plot of motor running

### Reaction Wheel with Stabilize Data Processing

The basic simulation functions for the Kalman filter are shown here. Several initial data points are needed, including the mass of the reaction wheel, the acceleration due to gravity, the mass of the bike, the motor's specifications, and the position of the bike's center of gravity. I can calculate the dynamic model elements, including the overall mass of each of the aforementioned components, using all of this data. The dynamic model matrices, which also include state-space matrices, must be calculated next. See Figure 11, the simulation program will produce a result that demonstrates how the Kalman filter is lowering the noisy vibration using all the calculated data.

Finally, see Figure 12, the Kalman filter below show a decent result. The left top graph of Roll Angle/Time shows Kalman filter simulation results. When the balancing wheel is triggered after 10 seconds, roll angle creates a significant floating change before tending to level out once more in less than three seconds, as shown in the top right graph, Roll Rate/Time. The duty cycle has a considerably more frequent % change in the first 10 seconds, according to the bottom right graph, which matches the Roll Rate/Time data result. Finally, the bottom right graph shows that the trend of

the response wheel RPM is comparable to the Kalman Roll Angle and kindly reflects the initial data finding.

```
clc, clf, clear all, close all

% Bike's Specifications
m_bike = 16.3; % Mass of the bike, kg
l_bike = 0.399; % Bike's CoG position, m
J_bike = 0.469954; % Mass moment of Inertia, Kg.m^2

% Reaction Wheel's Specifications
m_rw = 1.679; % Mass of the reaction wheel, kg
J_rw = 2.359e-3; % Mass moment of Inertia, kg.m^2

% Motor's specs
m_stat = 0.484; % Mass of the stator, kg
J_stat = 2.1e-4; % Mass moment of inertia of stator, kg.m^2
m_rot = 0.100; % Mass of the rotor, kg
J_rot = 9.394e-5; % Mass moment of inertia of rotor, kg.m^2

Rm = 19.7e-3; % Motor resistance, Ohm
KV = 140; % BLDC Velocity Constant, RPM/Volt
Kt = (3/2)*(1/sqrt(3))*(30/pi)*(1/KV); % Torque Constant, Nm/A
Kb = Kt; % Back EMF Constant, V/Rad/s

bm = 0.0001; % Friction coeff. of motor, N m/s /rad

% Design spec of the motor and reaction wheel
l2 = 0.580; % Position of the rw respected to ground, m

% Environment
g = 9.80; % Acceleration due to gravity, m/s^2

% Calculate the dynamic model elements
m1 = m_bike + m_stat; % Total mass of the bike and motor's stator, kg
l1 = (m_bike*l_bike + m_stat*l2)/(m1); % Position of new CoG, m
J1 = J_bike + J_stat + m_stat*(l2 - l_bike)^2; % Total mass moment of inertia, kg.m^2

m2 = m_rot + m_rw; % Total mass of reaction wheel and motor rotor, kg
J2 = J_rot + J_rw; % Total mass moment of inertia of rw and motor rotor, kg.m^2
```

```
% Calculate the dynamic model matrices
M = [1 0 0;
     0 m1*l1^2 + m2*l2^2 + J1 + J2 J2;
     0 J2 J2];

V = [0 -1 0;
     -(m1*l1 + m2*l2)*g 0 0;
     0 0 bm - Kt*Kb/Rm];

S = [0 0 Kt/Rm]';
```

```
% Calculate the state-space matrices
A = -inv(M)*V % A-matrix
```

```
A = 3x3
      0      1.0000      0
    20.0161      0    -0.0463
   -20.0161      0    72.2151
```

```
B = inv(M)*S % B-matrix
```

```
B = 3x1
    10^3 ×
      0
   -0.0008
    1.2232
```

```

t0 = 0; % Initial Condition, sec
t_run = 6; % Runing time, sec
dt = 0.01; % Sampling time, sec

% Initial condition
theta(1) = 2; % Theta, deg
theta_dot(1) = 0; % Theta_dot, RPM
wheel_speed(1) = 0; % Reaction Wheel speed, RPM

X(:,1) = [theta(1)*pi/180; theta_dot*pi/30; wheel_speed*pi/30];

Vm(1) = -K*X(:,1);

% Simulation program
n = t_run/dt;
t(1) = t0;
for i = 2:n
    % Update the time
    t(i) = t(i-1) + dt;

    % Calculate the voltage signal for the BLDC motor
    Vm(i) = -K*X(:,i-1);
    % if Vm(i) >= 22.2 Vm(i) = 22.2; end
    % if Vm(i) <= -22.2 Vm(i) = -22.2; end

    % Calculate the states of the whole system (bike and the rw)
    X(:,i) = X(:,i-1) + dt*(A*X(:,i-1) + B*Vm(i));
end

% Extract the results
theta = (180/pi)*X(1,:);
theta_dot = (30/pi)*X(2,:);
wheel_speed = (30/pi)*X(3,:);
T_rwp = (Kt/Rm)*Vm - (Kt*Kb/Rm)*(wheel_speed*pi/30);

% Plot the response and the torque
figure;
subplot(2, 2, 1); plot(t, theta, 'k-', "linewidth", 1.5);
xlabel("Time (sec)"), ylabel("Theta (deg)"), grid;

subplot(2, 2, 2); plot(t, T_rwp, 'g-', "linewidth", 1.5);
xlabel("Time (sec)"), ylabel("Torque (Nm)"), grid;

subplot(2, 2, 3); plot(t, wheel_speed/1000, 'r--', "linewidth", 1.5);
xlabel("Time (sec)"), ylabel("IWP Speed (1000 RPM)"), grid;

subplot(2, 2, 4); plot(t, Vm, 'r-', "Linewidth", 1.5);
xlabel("Time (sec)"), ylabel("Voltage (Volts)"), grid;

```

Figure 11. MATLAB script of Kalman filter solution applies on reaction wheel

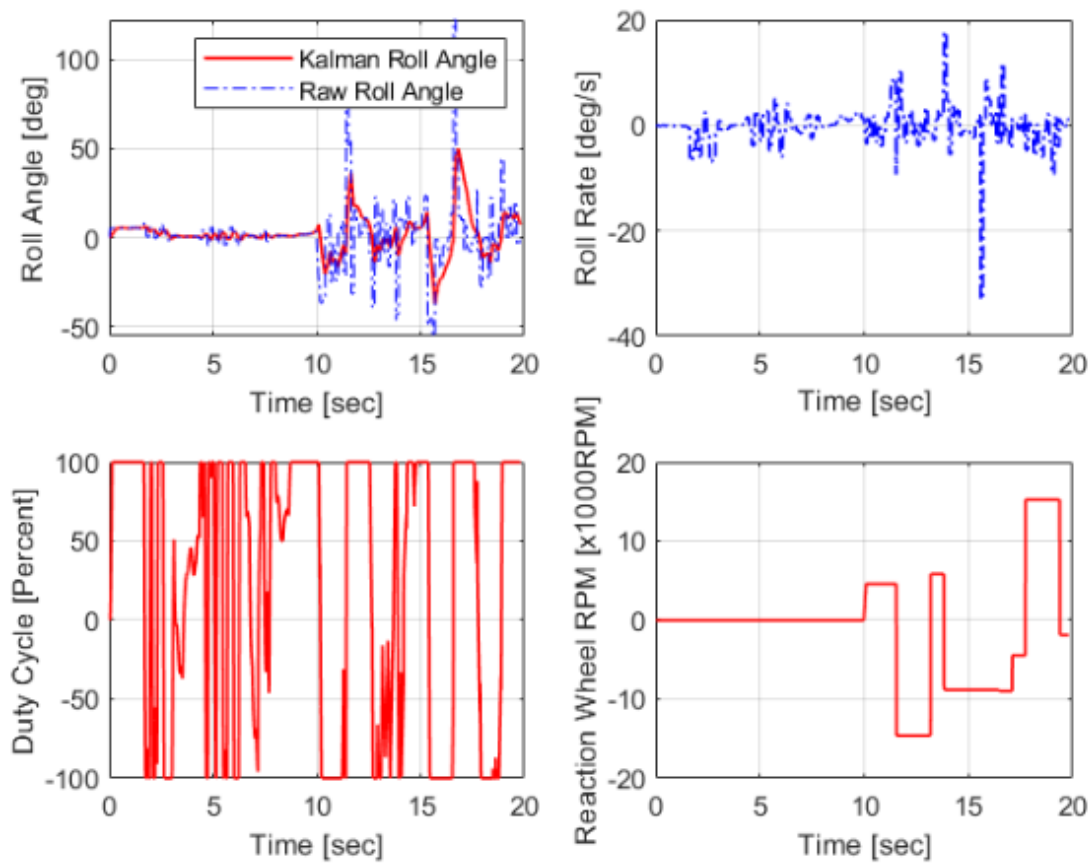


Figure 12. Simulation plots show the time vs. Roll Angle, Roll Rate, Duty Cycle and Reaction Wheel RPM with a 2-way Filter Solution

## CHAPTER 4

### IMAGE PROCESSING BASED ON MACHINE LEARNING FOR SELF-DRIVING BICYCLE

After establishing the bicycle's skeleton, I need now enable the bicycle to see and think. The lane process and object detection parts make up most of my code. To fulfill the goals in these two parts, I used Raspberry Pi, OpenCV, and TPU (Tensor Process Unit).

#### Lane Processing

Perception (lane detection) and Path/Motion Planning are the two components of a lane keep assist system (steering). Lane detection's task is to convert the coordinates of identified lane lines from a video of the road. One approach to do this is to use the computer vision library OpenCV, which I installed. However, we must first be able to recognize lane lines in a single image before we can detect them in a video.

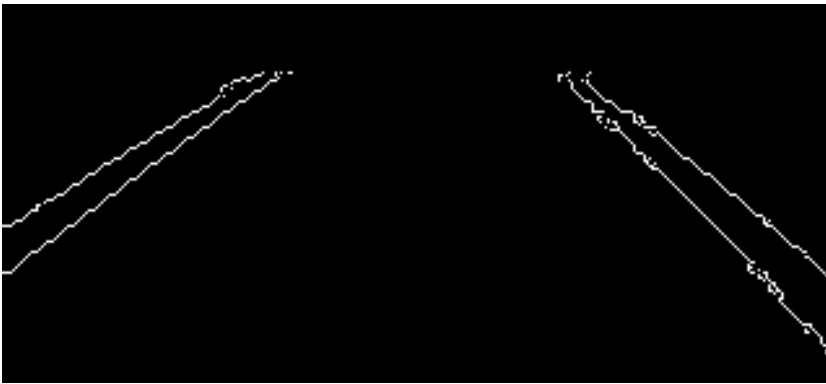
For computer vision work, I used Python and OpenCV. The term "open-source computer vision" stands for "open-source computer vision." You can use OpenCV's extensive collection of functions. The OpenCV library has a lot of documentation.

The Canny edge detection command is a potent tool for detecting picture edges. The image is first turned to grayscale. Each pixel has a gray value between 0 and 255, and the color of the lane is usually distinct from that of the road surface. For detection, we can apply color mutation from the road surface to the lane surface. Calculate the gradient after that. The strength of the point's gradient correlates to the brightness of each pixel; edges are created by tracing the pixels under the maximum gradient. The edges of the object are used to determine its form.

The shape of the thing is determined by sensing the object's boundaries. The parameters for the OpenCV Canny function are as follows: `edges=cv2.Canny(gray, low_threshold, higy_threshold)`

Gray scale map as input, border map as output. The strength of the boundary to be identified is determined using low and high thresholds. The difference in the values of neighboring pixels in the image can be used to calculate the edge's intensity. It's the level of grayness. I see bright spots, black areas, and all the gray in between when I look at a grayscale image. See Figure 13, the edge is

defined by a quick change in brightness. Because my graph is just a mathematical function of  $x$  and  $y$ , we can use it to perform arithmetic operations on it just like any other function.



*Figure 13. Edges of the lanes*

We could, for example, take its derivative, which is simply a measure of the function's change. A minor derivation entails a minor alteration. The bigger change, bigger derivation. Because the graph is two-dimensional, taking both derivatives with respect to  $x$  and  $y$  makes logical. This is known as gradient, and it is calculated by measuring how quickly the pixel values of each point in the image change, as well as in which direction they change the most quickly.

When I calculate gradients, result gets thick edges. I use the Canny method to discover each pixel after the strongest gradient by streamlining these edges. The stronger edges are then extended to include pixels up to the lower threshold given by the Canny function. We already have an 8-bit image after converting to grayscale, thus each pixel may take  $2^8 = 256$  different values. Pixel values range from 0 to 255 as a result. The derivative (basically, the difference in value between each pixel) will be in the tens or hundreds in this range.

One lane line in the image: In most cases, the camera should be able to view both lane lines. However, there are moments when the automobile begins to drift out of the lane, either due to faulty steering logic or a steep bend in the lane. The camera can only capture one lane line at this time. As a result,  $\text{len}(\text{right fit}) > 0$  and  $\text{len}(\text{left fit}) > 0$  must be checked in the code above.

Vertical line segments: As the car turns, vertical line segments are occasionally recognized. We can't average the slopes of vertical lines with the slopes of other line segments, even though they



aren't erroneous detections. I choose to overlook them for the sake of simplicity. Because vertical lines are uncommon, they have little effect on the lane recognition algorithm's overall performance. Alternately, the image's X and Y coordinates may be flipped, resulting in vertical lines with a slope of zero, which could be included in the average. However, see Figure 14, since the camera is usually oriented in the same direction as the lane lines, rather than perpendicular to them, the horizontal line segments would have an infinite slope. Another option is to represent the line segments in polar coordinates, then average the angles and distances from the origin.



*Figure 14.* Live video demo on the path

Now that I know where the bicycle is going, I need to convert it to a steering angle so we can tell the automobile to turn. A steering angle of 90 degrees means you're going straight, 45–89 degrees mean you're turning left, and 91–135 degrees mean you're turning right on the bicycle I developed.

When I construct the model and print the parameters list, it displays that there are around 250,000 parameters, see Figure 15. This is a good way to ensure that each layer of my model is produced according to my expectations.

```
Please use rate instead of keep_prob . Rate should be set to rate = 1 - keep_prob .
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 20, 64)	27712
dropout_1 (Dropout)	(None, 3, 20, 64)	0
conv2d_5 (Conv2D)	(None, 1, 18, 64)	36928
flatten_1 (Flatten)	(None, 1152)	0
dropout_2 (Dropout)	(None, 1152)	0
dense_1 (Dense)	(None, 100)	115300
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

Total params: 252,219  
 Trainable params: 252,219  
 Non-trainable params: 0

None

Figure 15. Training model parameter list [28]

The  $R^2$  measure is another way to assess how well our model worked. See Figure 16, the model functions well if the  $R^2$  is near 100 percent. As you can see, even with 200 photographs, my model has an  $R^2$  of 93 percent, which is rather impressive, owing to the usage of image augmentation.

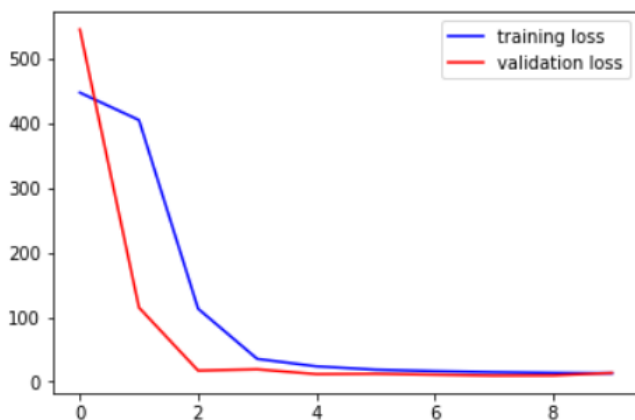


Figure 16. Training loss vs. Validation loss [29]

## Object Detection

Object identification, a practical application of deep learning algorithms, is the technology of utilizing a computer to scan, analyze, and comprehend images in order to recognize targets and objects of various patterns. There are four steps in the standard picture recognition process: image acquisition, image preprocessing, feature extraction, and image recognition are all steps in the image recognition process. The key features of an image are used to recognize it. Each image has its unique characteristics, such as the sharp point on the letter A, the circle on the letter P, and the acute angle at the center of the letter Y. The study of eye movement in image identification reveals that the line of sight is constantly focused on the image's primary elements, such as the location where the image's contour curvature is the largest or the contour direction changes abruptly, which contains the most information. The eye's scanning path is continually changing from one feature to the next. As a result, in the picture recognition process, the perceptual mechanism must filter out redundant input data and extract the essential data. Simultaneously, there must be a brain mechanism that integrates the information gathered in phases into a complete perceptual image. Artificial intelligence's field of image recognition is crucial.

In order to construct computer systems that simulate human image recognition activities, various image recognition models have been proposed. For instance, consider the template-matching model. According to this paradigm, in order to recognize an image, one must have the image's memory pattern from a previous experience, also known as a template. The image is recognized if the present stimulus fits the template in the brain. For example, if there is a template A in the brain, and the letter A's size, orientation, and shape are identical to the template A's, the letter A will be recognized. This model is straightforward and straightforward to use in practice.

However, this approach highlights that before a picture can be recognized, it must be perfectly congruent with the template in the brain. People can recognize both images that are perfectly consistent with the template in the brain and those that are not entirely consistent with the template. People can distinguish not only a single letter A, but also the letter A printed, handwritten,

misdirected, and in various sizes. See Figure 17, people can recognize a vast number of images at the same time; however it is impossible if each image has a corresponding template in the brain. Image segmentation can be done in a variety of ways, including threshold segmentation, edge detection, region extraction, and segmentation using theoretical tools. Gray image segmentation, color image segmentation, and texture image segmentation are the three types of image segmentation. The edge detection operator was proposed in 1965 [21], and it spawned a slew of classic edge detection methods.



*Figure 17. Trained imaged dataset detection and prediction*

However, with the rapid growth of image segmentation based on histogram and wavelet transforms, as well as VLSI technology, image processing research has advanced significantly in the last 20 years. Image segmentation approaches include a variety of theories, methodologies, and tools, including image segmentation based on mathematical morphology, wavelet transform segmentation, and genetic algorithm segmentation, among others.

## YoloV5

In the field of object detection, YOLO is a common deep learning-based detection approach. It has a good global receptive field, grid division, anchor frame matching, and a technique for detecting multi-semantic fusion [27]. YOLO model directly predicts the bounding box and probabilistic likelihood of picture objects with CNN, which dramatically improves detection accuracy when compared to classic object recognition methods.

Input, Backbone, Neck, and Head are all elements of the YOLOv5 basic model. The network's input section uses mosaic data augmentation, adaptive picture scaling, and adaptive anchor computation to process the image to be detected. The processed images are initially supplied into the focus structure in Backbone before being sliced, see Figure 18.

The focus accomplishment of a given task the input image information's integrity while also reducing the input size. After that, numerous Conv, C3 modules, and an SPP module extract features to create a new set of feature maps. A path aggregation network is used to aggregate the neck part of the network, which promotes feature fusion [27]. Bounding boxes are formed, and expected targets are categorized in the head section.

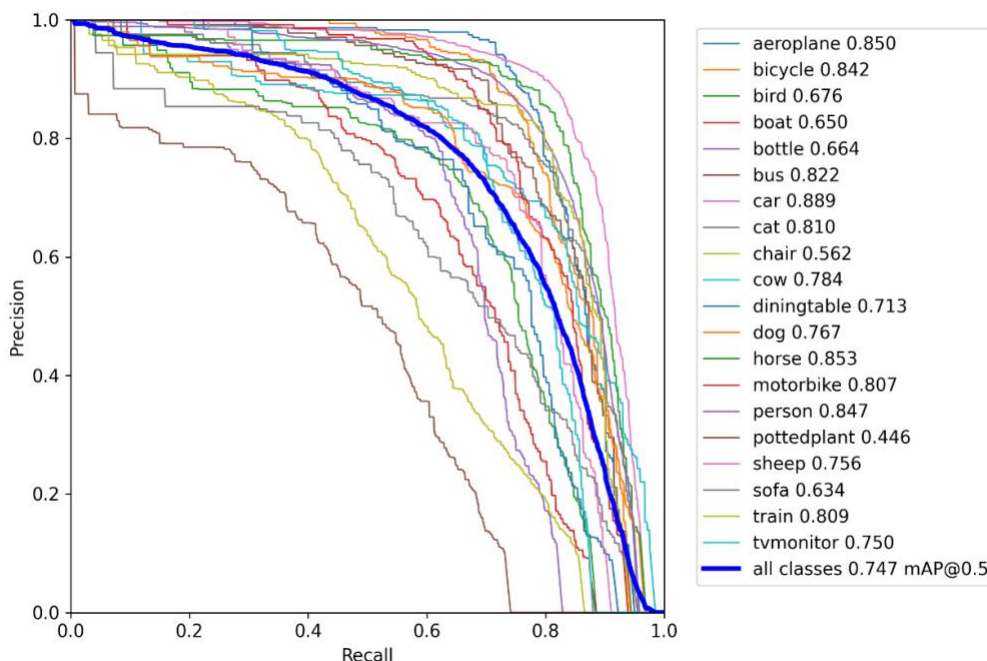
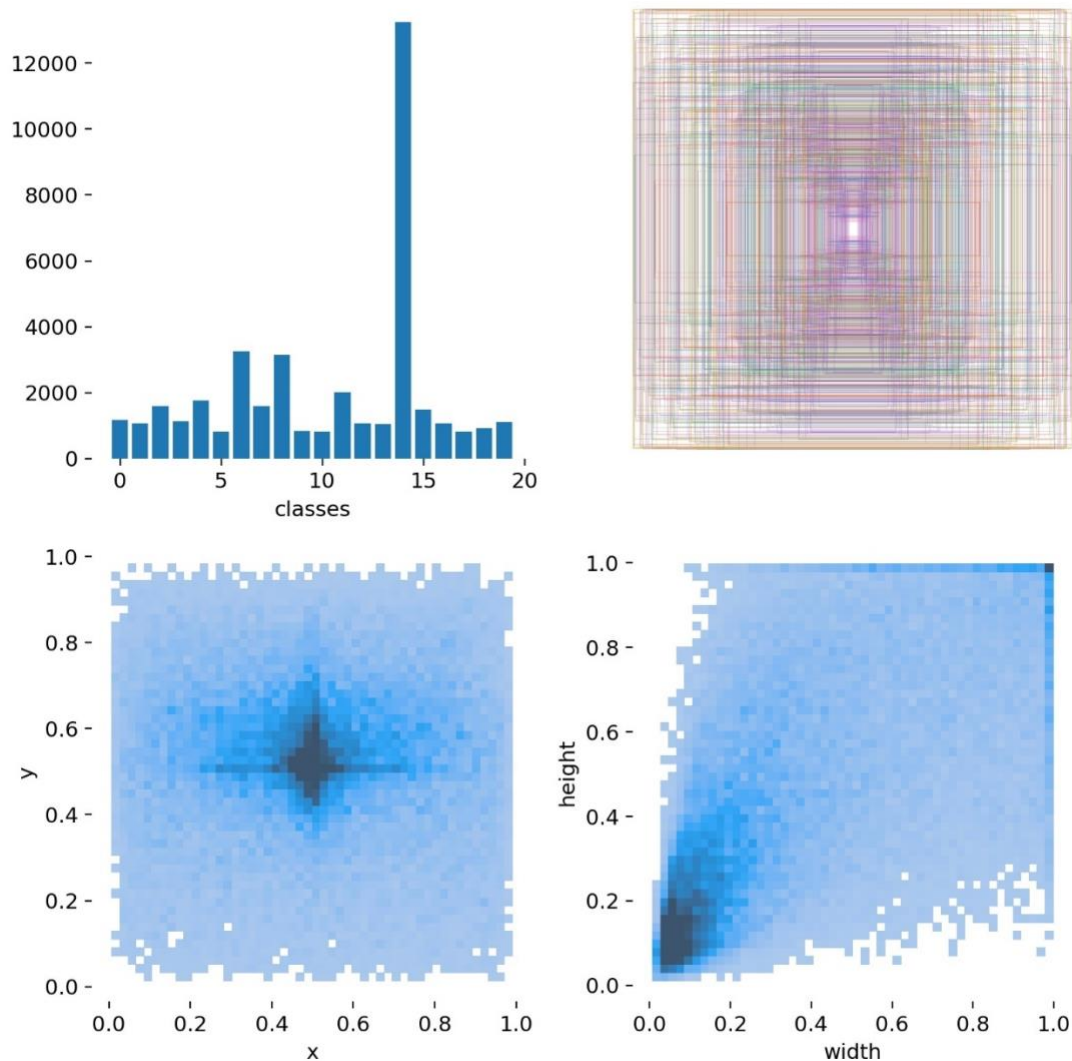


Figure 18. P.R. Curves (Precision-Recall Curves) [33]

The YOLO (You Only Look Once) algorithm was created to detect an object in real-time. To detect, the detection system employs a repurposed classifier or localizer. Starting with the YOLO version, YOLOv2, YOLOv3, and ending with YOLOv5, YOLO has several variants, see Figure 19 [28]-[30]. However, there has not been much work on applying the YOLOv5 algorithm to the library attendance system. As a result, it is critical to act.



*Figure 19.* The top right is the rectangles that are labeled (x,y,w,h) is the label of each instance label [33]

In the YOLO series, YOLOv4 and YOLOv5 were released. YOLOv5 is built on the PyTorch framework and has a detection speed of up to 140 frames per second. YOLOv5 is faster and more accurate than the preceding YOLO series, and its model is light and ideal for deployment on

embedded devices [29]. The same Mosaic data improvement technology as YOLOv4 is used in YOLOv5. The detection efficiency of small targets is increased by stitching the input images using random scaling, random cropping, and random layout, see Figure 20 [33].

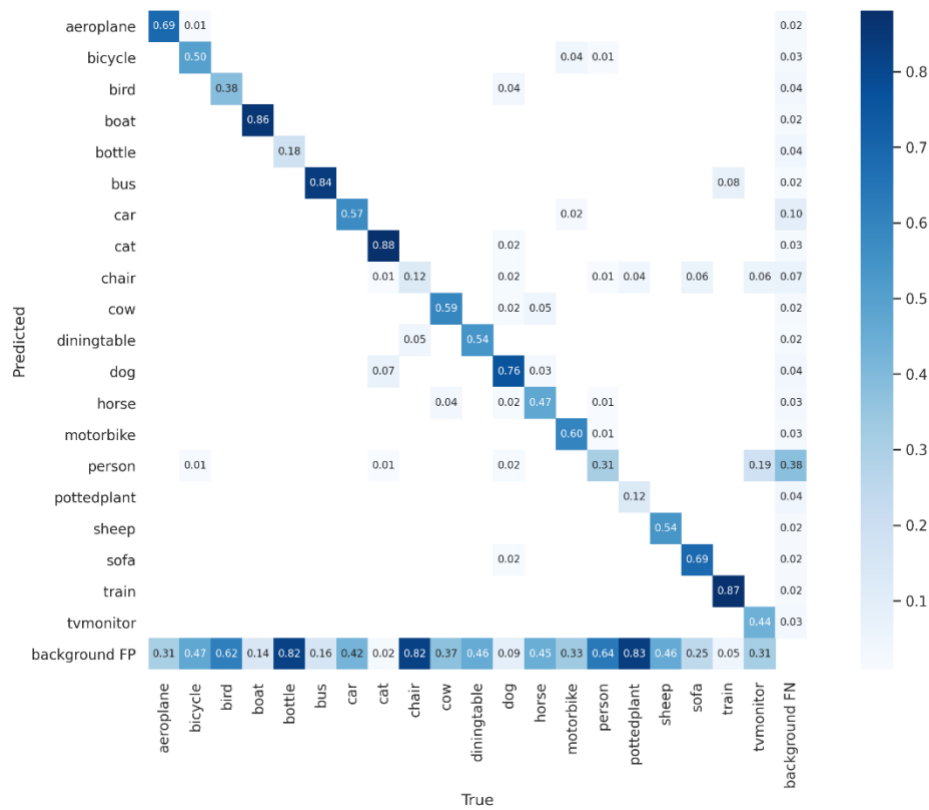


Figure 20. The plot of confusion matrix [33]

### Tensor Processing Unit (TPU)

At the end of the Google, I/O 2016 speech, Google mentioned a Tensor Processing Unit, or TPU, as one of the things they have done with A.I. and machine learning. The principle behind deep learning is to build layers, with the output of one layer feeding into the next. An input layer, numerous hidden layers, and an output layer make up a deep neural network. There are numerous neurons in each layer, and there are connection weights between them. Each neuron is modeled after a human nerve cell, while the connections between nodes are modeled after nerve cell connections. When compared to a CPU, the performance of a neural network may be considerably enhanced by employing a specific chip to speed up neuron processing. That's how Google's TPU is built, with a single command capable of doing several neuron calculations.



The difference between the `_edgetpu.tflite` file and the regular `tflite` file is that the `_edgetpu.tflite` file will perform all (99%) model inferences on the Edge TPU rather than the Pi's CPU, see Figure 21. In practice, this implies that the Edge TPU can process roughly 20 320x240 resolution photos per second (called FPS, Frames Per Second), but the Pi CPU alone can only handle about 1 FPS, see Figure 22. Twenty frames per second are (almost) real-time.

```
import pycoral.adapters.classify
import pycoral.adapters.detect
from pycoral.adapters import common
from pycoral.utils.edgetpu import make_interpreter
# from tflite_runtime.interpreter import load_delegate
from pycoral.utils.dataset import read_label_file
from PIL import Image
from traffic_objects import *
```

Figure 21. TPU required the PYCPRAL library to activate

```
def detect_objects(self, frame):
    logging.debug('Detecting objects...')

    # call tpu for inference
    start_ms = time.time()
    frame_RGB = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    image = Image.fromarray(frame_RGB)
    _, scale = common.set_resized_input(
        self.engine, image.size, lambda size: image.resize(size, Image.ANTIALIAS))
    self.engine.invoke()
    objects = self.engine.get_tensor(output_index)
    objects = pycoral.adapters.detect.get_objects(self.engine,
        score_threshold=self.min_confidence, image_scale=scale)
```

Figure 22. TPU functions required to run through the code



## CHAPTER 5

### HARDWARE IMPROVEMENT FPGA

The programmable Logic component of a typical FPGA, termed P.L., is the only part that can be changed (Programmable Logic). The Zynq series, on the other hand, is a single FPGA device with one or more ARM cores. PS refers to the SOC component (Processing System) [15][18][19]. In fact, you can build SOC directly on P.L., such as using ARM's Cortex-M3 kernel I.P., Micro Blaze. However, compared with directly solidified and embedded a mature ARM kernel circuit, it still consumes too many on-chip resources to build a kernel with similar performance to P.L. Since there are two parts on a chip that needs to be programmed, P.L. needs to describe the hardware architecture as a bitstream, and P.S. needs to run a standalone or operating system (such as Linux) [16], [17], [22]. This involves the problem of developing two areas separately, see Figure 23.

Those who are familiar with Pure P.L. devices such as Spartan, Kintex, and Virtex know that P.L. parts can be programmed, debugged, and run using Verilog language in Vivado software of Silas [20]. Those of you who have played with Zynq also know that you can add P.S. I.P. cores when you choose to use Zynq chips in Vivado. Once the hardware is exported, you can write programs that run bare-metal or on the operating system in the Silas SDK [19], [22]. You can also use Petalinux to load a hardware description file and generate a Linux operating system that matches your current hardware.

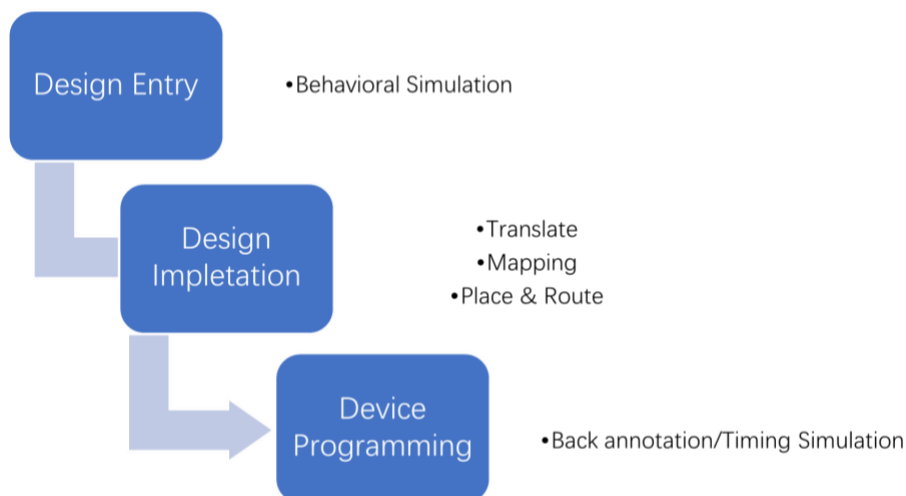


Figure 23. Traditional FPGA application flowchart

Because I am aware of the benefits and capabilities of FPGA, which include, first and foremost, increased parallelism, concurrency, and pipelines are used to accomplish this. (A): Concurrency is the process of allocating computing resources repeatedly such that many modules can run at the same time. This is comparable to multicore and SIMD technology now in use. However, unlike SIMD technology, FPGA concurrency can occur across multiple logic functions rather than being confined to the execution of the same function at the same time. For example, SIMD allows you to perform several adds simultaneously, but an FPGA can perform multiple additions, multiplications, and whatever logic you can imagine [20]. (B): Pipelining involves splitting jobs down into pieces and running them all at the same time. In truth, it is comparable to a CPU, except that CPUs deal with flow between instructions, whereas FPGAs deal with flow between jobs or threads. Second, it is adaptable. FPGA internal logic, known as Lookup Table logic, can be thought of as a hardware circuit [22]. The term "customizable" refers to the ability of users to create their own logic circuits when resources permit. Tasks execute quicker on hardware circuits than on software circuits in most cases. For example, in a CPU, two area instructions, two-bit and instructions, one shift instruction, one compare instruction, and one write back instruction are required to compare the size of 32 bits higher and 32 bits lower than a 64 bit, however with an FPGA, just one comparator is required. Finally, it is possible to alter it. The term "reconfigurable" refers to the ability to modify the logic inside the FPGA to meet specific needs, lowering development costs. At the same time, reusing resources using FPGA saves the server more space than utilizing numerous fixed ASIC modules [17][18]. To summarize, the next stage is to move all prior engineering to FPGA, which will ensure great efficiency and low power consumption. Below is an example of previous research of neural network application on FPGA in the traditional way that experimental hardware measurement is based on FPGA evaluation board Xilinx Ultrascale+ KCU116 with the XCKU5P computing unit being connected to a P.C. through PCI Express.

The PCI Express allows the FPGA to access the DDR4 ram as temporary storage for the input data and model [15][19][22]. The LUT and Flip-Flop (F.F.) resources are demanded to process the

on-chip computation. Off-chip memories are also required for our design because it is not realistic to store the model in the on-chip register only. The F.F. is used as the shared register and buffer, while the piped-lining rate is not possible to raise due to the dependency of the data. The F.F. utilization is 20-30% and varies with the model compressing ratio. The BRAM utility remains the same because of using tensor shape manipulation that is designated to hold more space for redundancy and not tightly allocate the data and model. To process the convolution, I designed MAC (Multiply Accumulate) as processing units that consist of multiple DSPs. I observed that the higher the parallelism channel lane deployed, the higher DSP utilization will be obtained [20]. See Figure 24. To raise the speed of the process, I can invest more DSP resources for the design. I measured the power consumption within two places: on-chip and off-chip. The on-chip power is drained by all on-chip resources and the connection between the chip and the I/O supply by the evaluation board. Off-chip power is consumed by the DDR4 memory that is connected through the PCI-E to the FPGA [22].

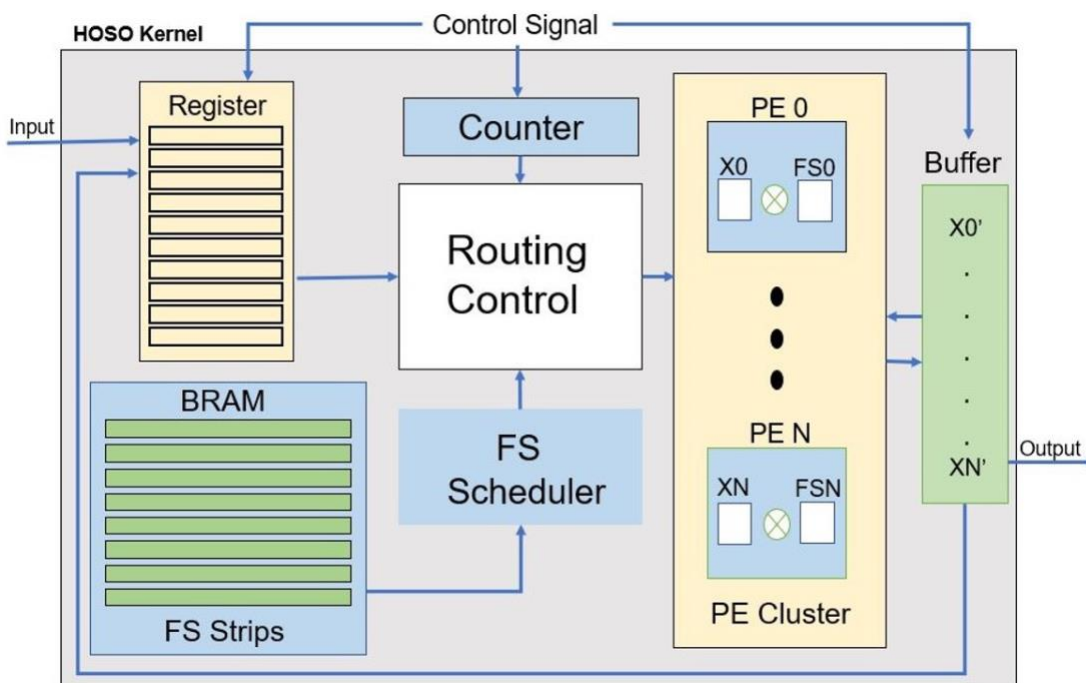


Figure 24. Schematic design of traditional FPGA method from previous research

While porting the engineering, however, I discovered a more convenient and efficient FPGA performance. This new feature may be used to design not just standard applications but also other

neural network models that are now available on the market. VITIS AI, VITIS, PETALINUX, and DPU are all used in the new technique.

### **Vitis IDE Instruction**

For Vitis, an important concept is a platform, which is not just a description of the hardware derived from Vivado, but a more general concept, which includes the concept of hardware and software integration.[23] On this platform, you can perform many comprehensive operations. The hardware can be exported using Vivado. See Figure 25, using Vivado after installing Vitis, you can either invoke the Vitis IDE directly after compiling the hardware part or create a new project from the Vitis IDE and choose to use the existing platform that you have added. Or select the XSA (Xilinx Shell Archive) file exported from Vivado to create a new platform. The XSA file is exported by Vivado and is the abbreviation of Xilinx Shell Archive [24][25]. It mainly contains the hardware information of the platform and is used as the input of the Vitis platform Project. The platform is a more complex and generalized concept that includes hardware and software and can support different design processes such as hardware acceleration and embedded design. In the software part, we can also write bare-metal or operating system applications directly in Vitis IDE and compile and debug them. In this way, the Vitis IDE itself is actually very similar to previous SDKs.

### **Vitis A.I.**

Vitis A.I. is currently command-line only and runs only on Linux, meaning that in order to use this feature, you must have a Linux version of Vitis installed and a large enough running resource. As shown in the overview, The Vitis A.I. can load either hardened models from the Model Zoo on Github, or customized Model files from users. The Vitis A.I. tool consists of several components, see Figure 26, an A.I. compiler, quantizer, optimizer, analyzer, and deploys the final model on the Deep Learning Processing Unit (DPU) on the P.L. side. The P.S. side, or in the case of the Alevo accelerator card, the P.C. side, can be called through the Xilinx Runtime Library (XRT) interface [23]-[26].

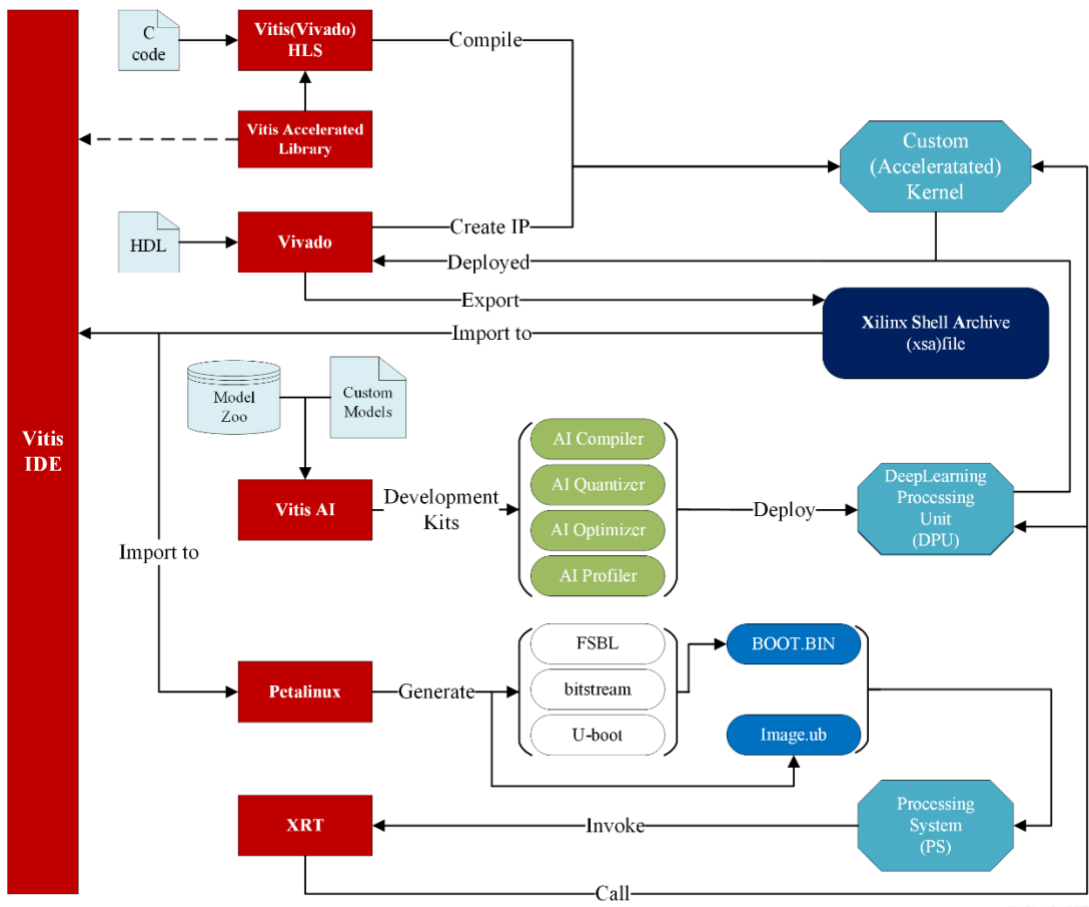


Figure 25. Overview Platform of VITIS [26]

### Deep-learning Processor Unit (DPU)

The DPU (deep-learning processor unit) is a programmable engine designed specifically for deep neural networks. It is a set of parameterizable I.P. cores that come pre-installed on the hardware and don't need to be routed. It's made to help deep learning inference algorithms, which are commonly used in computer vision applications, including image/video classification, semantic segmentation, and object detection/tracking, run faster. The Vitis A.I. specialized instruction set is included with the DPU, allowing for the efficient development of deep learning networks. Various prominent convolutional neural networks, including VGG, ResNet, GoogLeNet, YOLO, SSD, and MobileNet, are supported and accelerated using an efficient tensor-level instruction set [24], [25].

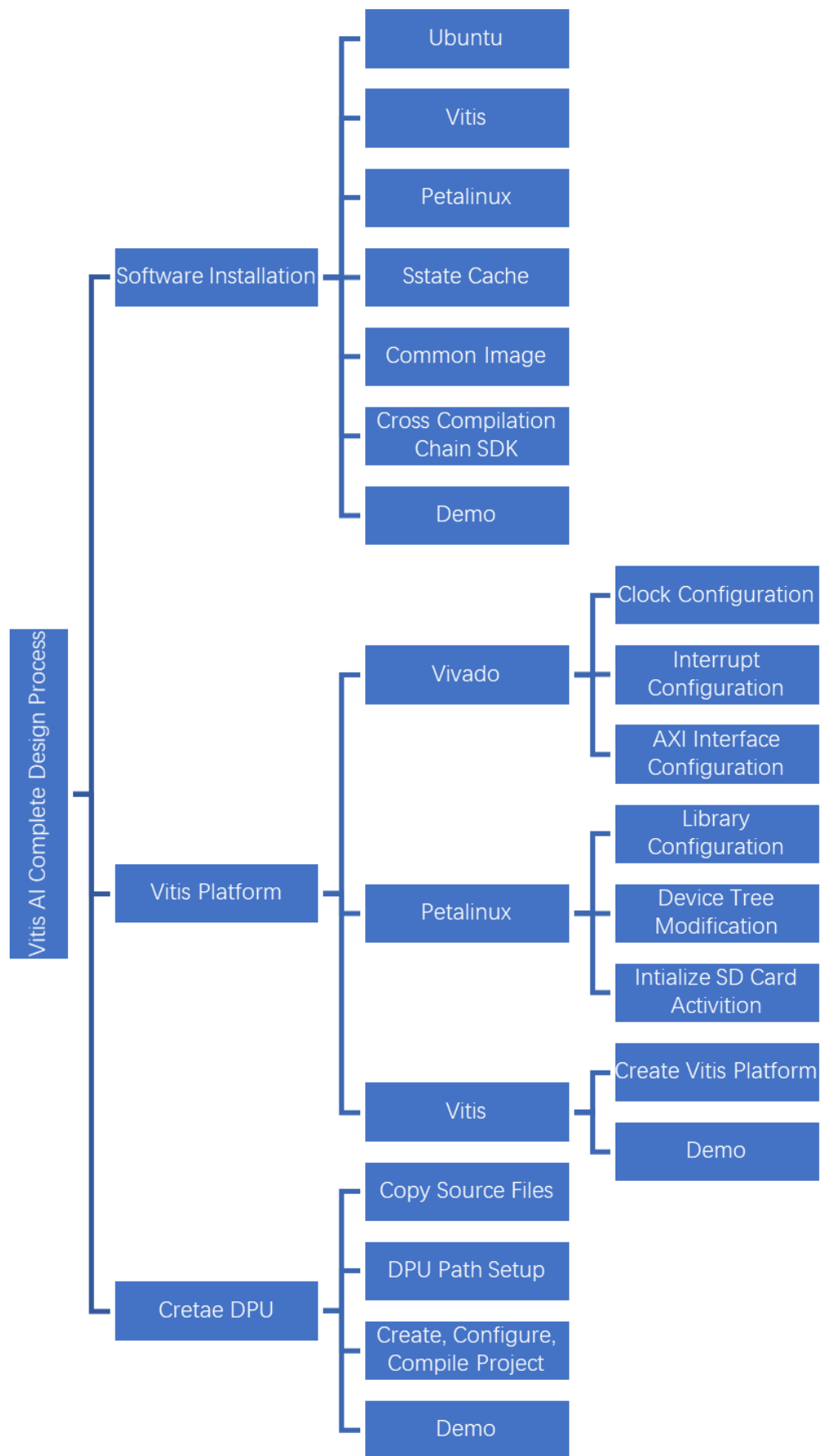


Figure 26. VITIS AI design process

## CHAPTER 6

### APPLICATION DEVELOPMENT AND PROCEDURES

After introducing the related software and concepts that will be used above, I will show the whole system process, which is mainly divided into hardware layer and software layer. The FPGA model I used was produced by XILINX company Zynq® UltraScale+™ MPSoC ZCU104. It helps designers quickly launch designs for automotive, industrial, video, and communications applications. It helps designers quickly launch designs for embedded vision applications such as surveillance, advanced driver assistance systems (ADAS), machine vision, augmented reality (A.R.), unmanned aerial vehicles, and medical imaging. Support for all major peripherals and interfaces for various application development. Very suitable for the above self-balancing cycling project.

The hardware layer mainly includes VIVADO to establish the corresponding FPGA hardware information, configure PETALINU, and finally establish the VITIS Platform. The purpose of VIVADO is to create a base project to tell VITIS which resources are available and which resources can be set within the platform. See Figure 27. When VITIS is called, it is designed to automatically connect to VIVIADO based on its I.D.

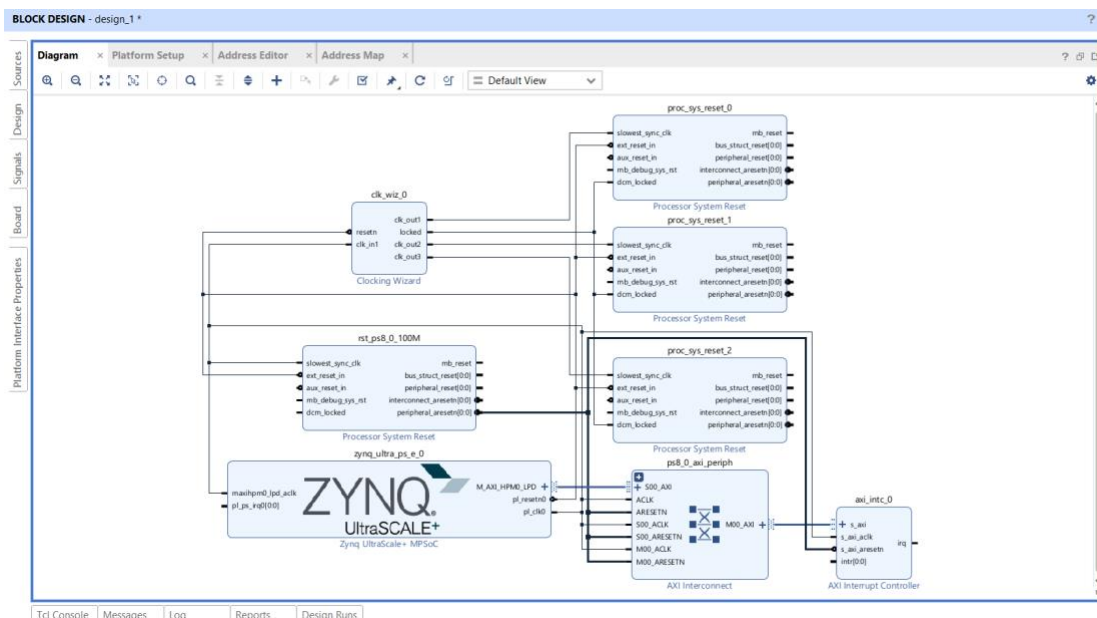


Figure 27. Basic VIVADO IP Core Design with Zynq UltraScale+ MPSoC

Developers may use PetaLinux tools to synchronize the software platform with the hardware design when new features and devices are added. See Figure 28 and Figure 29, the PetaLinux tools will create a bespoke Linux Board Support Package for you, complete with device drivers for Xilinx embedded processor I.P. cores, kernel, and boot loader settings. Software engineers may concentrate on their value-added applications rather than low-level development duties with this capability.

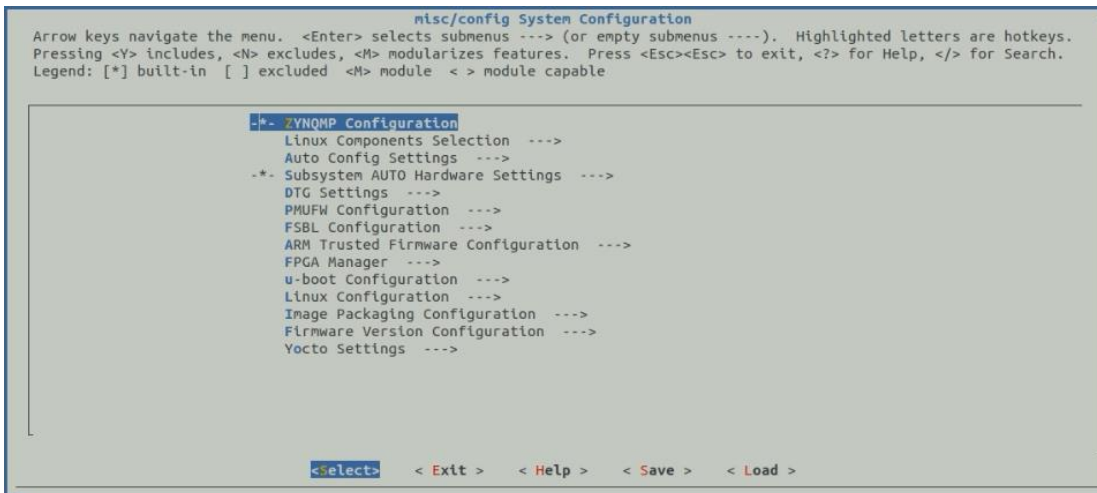


Figure 28. Petalinux Configuration screenshot

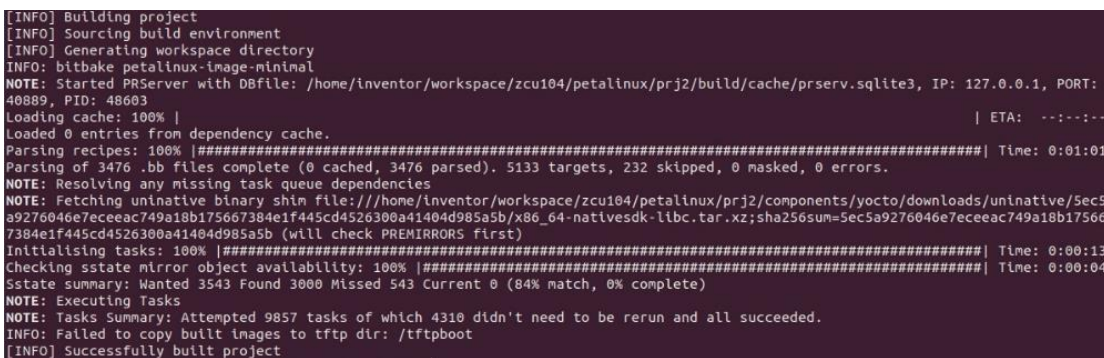


Figure 29. Petalinux generation process screenshot

On the software layer, VITIS AI generates a DPU kernel through training, quantification and compilation of the neural network model and imports it into the previously established DPU platform so that the actual effect of the neural network model can be realized on FPGA, see Figure 30.



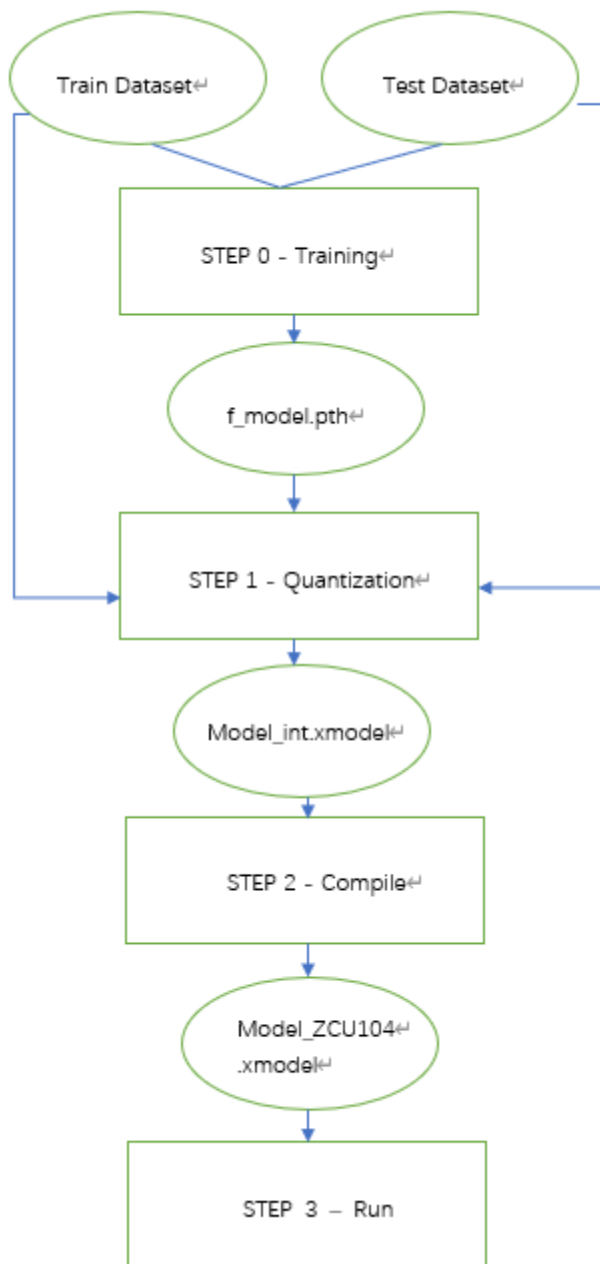


Figure 30. Flowchart of VITIS AI procedure

1. Prepare a trained model. This step is a common A.I. model. After the training, pruning is required, and it will be very slow for YOLOV5 without pruning. See Figure 31. Therefore, what I did was to put the trained model into VITIS AI and prune again. Because we have a post-training model ready, so we don't have to retrain.
2. Quantification based on the neural network that has been used to quantize the previous .pth file to the .xmodel file. See Figure 32. After the first quantization, the model has 98.75% accuracy.

```

rand_in = torch.randn([batchsize, 1, 28, 28])
quantizer = torch_quantizer(quant_model, model, (rand_in), output_dir=quant_model)
quantized_model = quantizer.quant_model

```

Figure 31. VITIS AI main quantification functions

```

[VAIQ_WARN]: CUDA is not available, change device to CPU
[VAIQ_NOTE]: Quantization calibration process start up...
[VAIQ_NOTE]: =>Quant Module is in 'cpu'.
[VAIQ_NOTE]: =>Parsing CNN...
[VAIQ_NOTE]: Start to trace model...
[VAIQ_NOTE]: Finish tracing.
[VAIQ_NOTE]: Processing ops...
| 13/13 [00:00<00:00, 695.54it/s, OpInfo: name = CNN/Sequential[network]/Flatten[1
[VAIQ_NOTE]: =>Doing weights equalization...
[VAIQ_NOTE]: =>Quantizable module is generated.(build/quant_model/CNN.py)
[VAIQ_NOTE]: =>Get module with quantization.
Test set: Accuracy: 9875/10000 (98.75%)

```

Figure 32. Screenshot of VITIS AI quantification process

3. Compile. After generating the xmodel file now needs to compile with the hardware file we prepared before in the hardware section from VIVADO together generate to a new xmodel, see Figure 33, and this one is good to use now because it also contains FPGA hardware fingerprint information (arch.json)

```

For TensorFlow 1.15 Workflows do:
  conda activate vitis-ai-tensorflow
For Caffe Workflows do:
  conda activate vitis-ai-caffe
For PyTorch Workflows do:
  conda activate vitis-ai-pytorch
For TensorFlow 2.6 Workflows do:
  conda activate vitis-ai-tensorflow2
Vitis-AI /workspace > conda activate vitis-ai-pytorch
(vitis-ai-pytorch) Vitis-AI /workspace > source compile.sh zcu104 build build/log
-----
COMPILING MODEL FOR ZCU104..
-----
tee: build/log/compile_zcu104.log: No such file or directory
[UNIOLOG][INFO] Compile mode: dpu
[UNIOLOG][INFO] Debug mode: function
[UNIOLOG][INFO] Target architecture: DPUCZDX8G_ISA0_B4096_MAX_BG2
[UNIOLOG][INFO] Graph name: CNN, with op num: 31
[UNIOLOG][INFO] Begin to compile...
[UNIOLOG][INFO] Total device subgraph number 3, DPU subgraph number 1
[UNIOLOG][INFO] Compile done.
[UNIOLOG][INFO] The meta json is saved to "/workspace/build/compiled_model/meta.json"
[UNIOLOG][INFO] The compiled xmodel is saved to "/workspace/build/compiled_model/CNN_zcu104.xmodel"
[UNIOLOG][INFO] The compiled xmodel's md5sum is 0f9719030949418ee93fdae1aea07850, and has been saved to "/workspace/build/compiled_model/md5sum.txt"

```

Figure 33. Screenshot of VITIS AI compile process

4. Integration. Before the final step, I need to create a VITIS platform first, also based on the VIVADO application I created in the hardware section. See Figure 34, Figure 35, and Figure 36. Finally, after I have prepared both the hardware information and software model, I need to import VITIS AI into VITIS first (otherwise will not generate the DPU core).

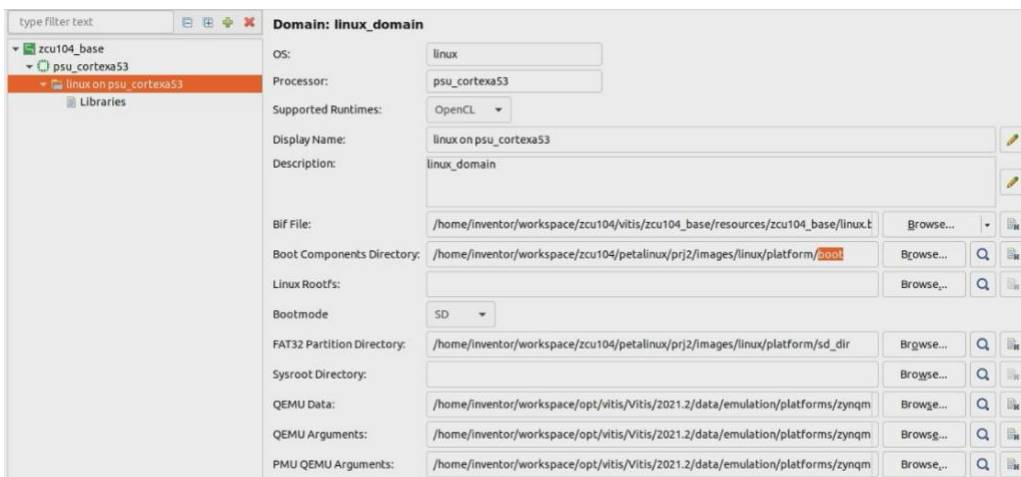


Figure 34. Establishment of VITIS platform (all directory paths are according to previous steps)

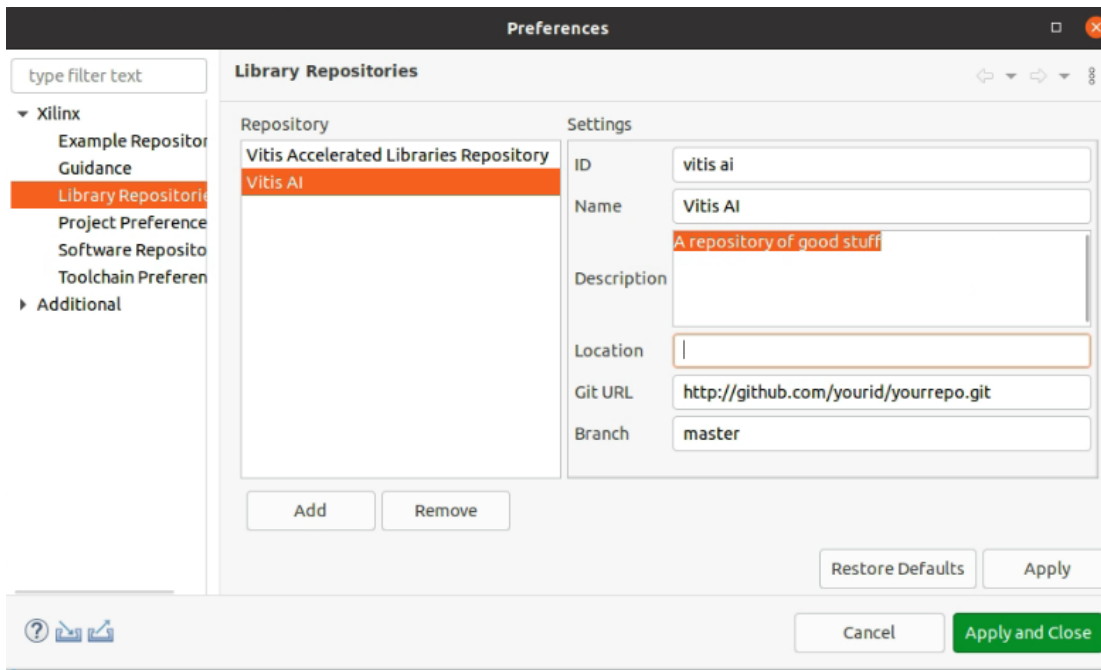


Figure 35. Import VITIS AI into the VITIS platform (most important step)

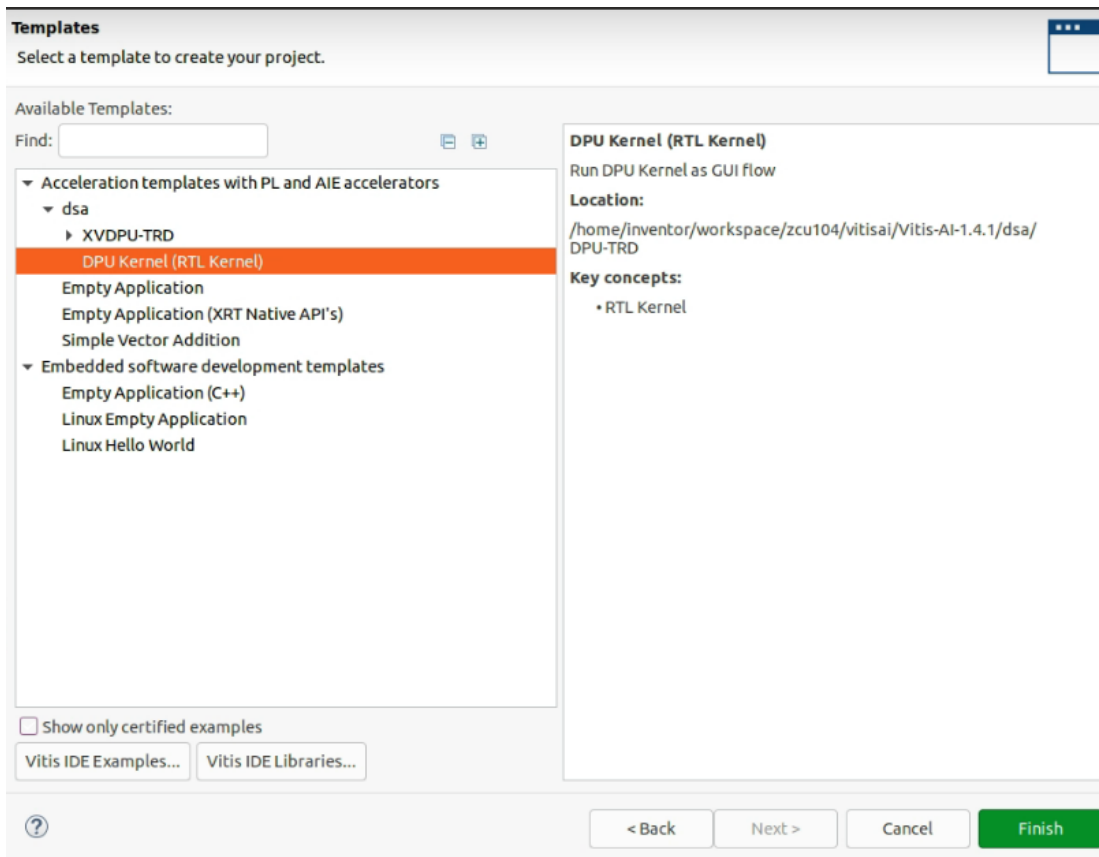


Figure 36. Establishment of DPU Core

## CHAPTER 7

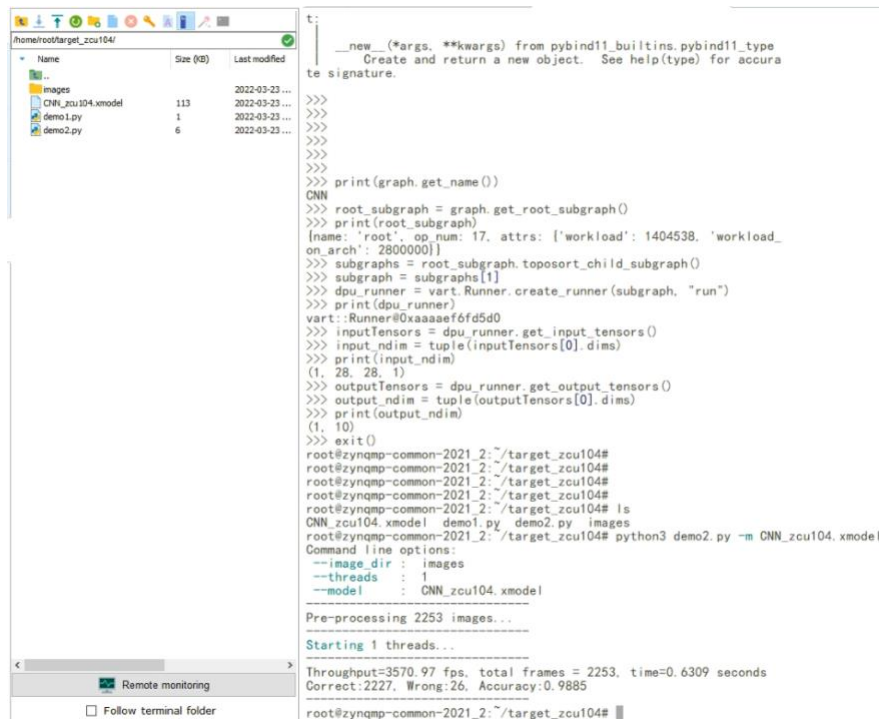
### APPLICATION RESULT

As a result, once I build the DPU core, I need to burn all the files from Petalinux, VITS AI, and VITIS into an S.D. card. Then boot the FPGA system from the S.D. card; therefore, I reach out that using FPGA just raspberry pi and Navida Xavier to run neural network model by command lines below are two examples: Resnet50 and Mnsit model. See Figure 37 and Figure 38. It is a pity that I was supposed to run a YOLOV5 model, but after all the research I did, there is no complete quantization of YOLOV5 now, so I can't generate xmodel file.

```
root@zynqmp-common-2021_2:~# ls
VART resnet50
root@zynqmp-common-2021_2:~# cd resnet50/
root@zynqmp-common-2021_2:~/resnet50# ls
001.jpg resnet50 resnet50.xmodel words.txt
root@zynqmp-common-2021_2:~/resnet50# chmod +x resnet50
root@zynqmp-common-2021_2:~/resnet50# ./resnet50 resnet50.xmodel
WARNING: Logging before InitGoogleLogging() is written to STDERR
I1001 23:36:11.993796 1158 main.cc:292] create running for subgraph: subgraph_conv1

Image : 001.jpg
top[0] prob = 0.982662 name = brain coral
top[1] prob = 0.008502 name = coral reef
top[2] prob = 0.006621 name = jackfruit, jak, jack
top[3] prob = 0.000543 name = puffer, pufferfish, blowfish, globefish
top[4] prob = 0.000330 name = eel
```

Figure 37. Resnet50 model running with DPU core (Probability reach 98%)



```
t:
__new__(*args, **kwargs) from pybind11_builtins.pybind11_type
Create and return a new object. See help(type) for accurate signature.

>>>
>>>
>>>
>>>
>>>
>>> print(graph.get_name())
CNN
>>> root_subgraph = graph.get_root_subgraph()
>>> print(root_subgraph)
[name: 'root', op num: 17, attrs: {'workload': 1404538, 'workload_on_arch': 2800000}]
>>> subgraphs = root_subgraph.toposort_child_subgraph()
>>> subgraph = subgraphs[1]
>>> dpu_runner = vart.Runner.create_runner(subgraph, "run")
>>> print(dpu_runner)
vart.Runner@0xa0a0e0f0d5d0
>>> inputTensors = dpu_runner.get_input_tensors()
>>> input_ndim = tuple(inputTensors[0].dims)
>>> print(input_ndim)
(1, 28, 28, 1)
>>> outputTensors = dpu_runner.get_output_tensors()
>>> output_ndim = tuple(outputTensors[0].dims)
>>> print(output_ndim)
(1, 10)
>>> exit()
root@zynqmp-common-2021_2:~/target_zcu104#
root@zynqmp-common-2021_2:~/target_zcu104#
root@zynqmp-common-2021_2:~/target_zcu104#
root@zynqmp-common-2021_2:~/target_zcu104#
root@zynqmp-common-2021_2:~/target_zcu104# ls
CNN_zcu104.xmodel demo1.py demo2.py images
root@zynqmp-common-2021_2:~/target_zcu104# python3 demo2.py -m CNN_zcu104.xmodel
Command line options:
--image_dir : images
--threads   : 1
--model     : CNN_zcu104.xmodel

Pre-processing 2253 images...
Starting 1 threads...

Throughput=3570.97 fps, total frames = 2253, time=0.6309 seconds
Correct:2227, Wrong:26, Accuracy:0.9885
root@zynqmp-common-2021_2:~/target_zcu104#
```

Figure 38. Mnist CNN model running with DPU core (Throughput = 3570.97fps, total frames = 2253, total time = 0.63 secs, Accuracy: 98.9%)

## CHAPTER 8

### CONCLUSION

To summarize, my contribution is I apply FPGA as a low device since most of the battery power needs to support the balancing wheel of the autonomous bicycle. On top of the image processing algorithm, I completed both software-hardware co-design regarding the existing machine learning model, as see in Table 1. Furthermore, investigating a new method of FPGA application configuration called Deep-learning Process Unit (DPU), in the above bicycle project, can maintain balance and navigation automatically. The whole achievement not only succeeded in realizing the bicycle by combining angular power conservation and induction motor to achieve single auto balance, but also in realizing the autonomous navigation technology by using neural network learning. The training and simulation of neural networks are expertly employed in the part of automatic navigation to build a relatively stable and mature model that can not only detect the status of roads without streaks with high resolution but also identify more than 98 percent of the difficulties, as see in Table 2. On top of that, thanks to a hardware upgrade, I could run a neural network model on an FPGA using my new method. Below, two tables briefly conclude my analysis of strength/weakness and software comparison of the new DPU block compared to the traditional FPGA method.

Table 1. Comparison of Deep Learning Hardware Acceleration Methods

Accelerated methods	Advantage	Disadvantage
CPU	A universal system structure, can work independently	Low computing efficiency, low energy efficiency ratio
GPU	Good parallel computing ability, general graphics processing	High energy consumption, high bandwidth requirements
FPGA	High flexibility, large design space, strong parallel ability	High design demand, high cost
DPU (This Article)	The high energy efficiency ratio, simple development process	Low flexibility, massive steps involved

Table 2. FPGA &amp; DPU Development and Software Tool Chain

Hardware	Traditional FPGA Develop	DPU (This Article)
Compile Language	Verilog/VHDL/C/C++	Python3
Compile Steps	Synthesis, I.P. Blocks	Train, Quantization, Compile
Compile Result	Bitstream	Xmodel, arch.json
Debug Method	Simulation	SD Card
Required Software	Vivado	Vivado, Vitis, Vitis AI, Petalinux

However, the project may have advanced further if a single FPGA had been used to control both the bike's auto-balancing and auto-navigation functions. This one, on the other hand, will be more focused on FPGA modularization and stratification, which is a wholly different notion. However, once this technology is implemented, we can anticipate that using only one FPGA for multi-functional applications at the same time will be very spectacular, and the power and energy savings will be unfathomable. Unfortunately, despite our knowledge of FPGA's extensive capabilities, mastering it will be challenging and time-consuming. It is not only because of the difficult framework of FPGA design and application but also because the development of neural networks is changing all the time, making hardware reality difficult to follow, which is why the summary of what involves the application of this new method in my paper is not extensive, causing some difficulty in my paper. As development prices decrease and applications become more widely available, FPGA will become the backbone of hardware in the field of neural network learning.

## REFERENCES

- [1] Block, Daniel J, Karl J. Astrom, and Mark W. Spong. Reaction Wheel Pendulum. San Rafael, CA: Morgan & Claypool, 2007. Print.
- [2] Montoya, O. Danilo, and W. Gil-González. "Nonlinear Analysis and Control of a Reaction Wheel Pendulum: Lyapunov-Based Approach." *Engineering Science and Technology, an International Journal*, vol. 23, no. 1, 2020, pp. 21–29.,
- [3] C. Huang, Y. Tung and T. Yeh, "Balancing control of a robot bicycle with uncertain center of gravity," 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017.
- [4] Yeh, T., Lu, H., & Tseng., P.H. (2019). Balancing Control of a Self-driving Bicycle. ICINCO.
- [5] He, J., Zhao, M. (2015). Control System Design of Self-balanced Bicycles by Control Moment Gyroscope. In: Deng, Z., Li, H. (eds) *Proceedings of the 2015 Chinese Intelligent Automation Conference*. Lecture Notes in Electrical Engineering, vol 338. Springer, Berlin, Heidelberg.
- [6] G. Belascuen and N. Aguilar, "Design, Modeling and Control of a Reaction Wheel Balanced Inverted Pendulum," 2018 IEEE Biennial Congress of Argentina (ARGENCON), 2018.
- [7] R. Olfati-Saber, "Global stabilization of a flat underactuated system: the inertia wheel pendulum," *Proceedings of the 40th IEEE Conference on Decision and Control* (Cat. No.01CH37228), 2001.
- [8] Grewal, Mohinder S, and Angus P. Andrews. *Kalman Filtering: Theory and Practice Using Matlab*. Hoboken, N.J.: Wiley, 2008. Internet resource.
- [9] Saunders, B., Sim, J., Kingstone, T. et al. *Saturation in qualitative research: exploring its conceptualization and operationalization*. (2018).
- [10] C. A. Merlo-Zapata et al., "Modeling and Construction of an Inertia Wheel Pendulum Test-Bed," 2014 International Conference on Mechatronics, Electronics and Automotive Engineering, 2014.
- [11] M. L. Hoang, M. Carratù, V. Paciello and A. Pietrosanto, "Noise Attenuation on IMU Measurement For Drone Balance by Sensor Fusion," 2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), 2021
- [12] Alfian, R. Ikhsan et al. "Noise Reduction in the Accelerometer and Gyroscope Sensor with the Kalman Filter Algorithm." (2021).
- [13] Spong, Mark W. et al. "Nonlinear control of the Reaction Wheel Pendulum." *Autom.* 37 (2001): 1845-1851.
- [14] Q. K. Ho and C. B. Pham, "Study on Inertia Wheel Pendulum Applied to Self-Balancing Electric Motorcycle," 2018 4th International Conference on Green Technology and Sustainable Development (GTSD), 2018
- [15] J. Han, Z. Li, W. Zheng and Y. Zhang, "Hardware implementation of spiking neural networks on FPGA," in *Tsinghua Science and Technology*, vol. 25, no. 4, pp. 479-486, Aug. 2020,

- [16] Yijun Liu, Yuehai Chen, Wujian Ye, Yu Gui, "FPGA-NHAP: A General FPGA-Based Neuromorphic Hardware Acceleration Platform With High Speed and Low Power", IEEE Transactions on Circuits and Systems I: Regular Papers, vol.69, no.6, pp.2553-2566, 2022.
- [17] Songsong Li, Lei Gong, Teng Wang, Chao Wang, Xuehai Zhou, "FEAS: A Faster Event-driven Accelerator Supporting Inhibitory Spiking Neural Network", 2021 12th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), pp.14-18, 2021.
- [18] Lizheng Liu, Deyu Wang, Yuning Wang, Anders Lansner, Ahmed Hemani, Yu Yang, Xiaoming Hu, Zhuo Zou, Lirong Zheng, "A FPGA-based Hardware Accelerator for Bayesian Confidence Propagation Neural Network", 2020 IEEE Nordic Circuits and Systems Conference (NorCAS), pp.1-6, 2020.
- [19] N. Shah, P. Chaudhari and K. Varghese, "Runtime Programmable and Memory Bandwidth Optimized FPGA-Based Coprocessor for Deep Convolutional Neural Network," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 12, pp. 5922-5934, Dec. 2018
- [20] Jingjing Si, Guoliang Li, Yinbo Cheng, Rui Zhang, Godwin Enemali, Chang Liu, "Hierarchical Temperature Imaging Using Pseudoinversed Convolutional Neural Network Aided TDLAS Tomography", IEEE Transactions on Instrumentation and Measurement, vol.70, pp.1-11, 2021
- [21] Injune Yeo, Sang-Gyun Gi, Gunuk Wang, Byung-Geun Lee, "A Hardware and Energy-Efficient Online Learning Neural Network with an RRAM Crossbar Array and Stochastic Neurons", IEEE Transactions on Industrial Electronics, vol.68, no.11, pp.11554-11564, 2021.
- [22] Muhammad Salman Ali, Tauhid Bin Iqbal, Kang-Ho Lee, Abdul Muqeet, Seunghyun Lee, Lokwon Kim, Sung-Ho Bae, "ERDNN: Error-Resilient Deep Neural Networks with a New Error Correction Layer and Piece-Wise Rectified Linear Unit", IEEE Access, vol.8, pp.158702-158711, 2020.
- [23] J. Wang and S. Gu, "FPGA Implementation of Object Detection Accelerator Based on Vitis-AI," 2021 11th International Conference on Information Science and Technology (ICIST), 2021
- [24] G. -Z. Lin, H. M. Nguyen, C. -C. Sun, P. -Y. Kuo and M. -H. Sheu, "A Novel Bird Detection and Identification based on DPU processor on PYNQ FPGA," 2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), 2021
- [25] J. Zhu, L. Wang, H. Liu, S. Tian, Q. Deng and J. Li, "An Efficient Task Assignment Framework to Accelerate DPU-Based Convolutional Neural Network Inference on FPGAs," in IEEE Access, vol. 8, pp. 83224-83237, 2020
- [26] S. Goel, M. Balakrishnan and R. Sen, "EnergyNN: Energy Estimation for Neural Network Inference Tasks on DPU," 2021 31st International Conference on Field-Programmable Logic and Applications (FPL), 2021
- [27] S. Luo, J. Yu, Y. Xi and X. Liao, "Aircraft Target Detection in Remote Sensing Images Based on Improved YOLOv5," in IEEE Access, vol. 10, pp. 5184-5192, 2022
- [28] H. Zhang, M. Tian, G. Shao, J. Cheng and J. Liu, "Target Detection of Forward-Looking Sonar Image Based on Improved YOLOv5," in IEEE Access, vol. 10, pp. 18023-18034, 2022



- [29] L. Li, M. Liu, L. Sun, Y. Li and N. Li, "ET-YOLOv5s: Toward Deep Identification of Students; in-Class Behaviors," in IEEE Access, vol. 10, pp. 44200-44211, 2022
- [30] Iza Sazanita Isa, Mohamed Syazwan Asyraf Rosli, Umi Kalsom Yusof, Mohd Ikmal Fitri Maruzuki, Siti Noraini Sulaiman, "Optimizing the Hyperparameter Tuning of YOLOv5 for Underwater Detection", IEEE Access, vol.10, pp.52818-52831, 2022.
- [31] Hartmann, Michael & Massimiani, Gianluca. (2018). State of the Art: Safety for pedestrians and autonomous vehicles (Technical Report).10.13140/RG.2.2.11639.27043.
- [33] Multiple full image bounding boxes receiving poor confidence scores #6903  
<https://github.com/ultralytics/yolov5/issues/6903>