



## Physical Aimbot สำหรับเกม FPS ด้วย Computer Vision

นายภณลภัส สุทธิมาลา รหัสนักศึกษา 65340500046

โครงงานนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร  
ปริญญาวิทยาศาสตรบัณฑิต สาขาวิชาวิศวกรรมหุ่นยนต์และระบบอัตโนมัติ  
สถาบันวิทยาการหุ่นยนต์ภาคสนาม  
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี  
ปีการศึกษา 2567

\*\*\* ไม่ต้องพิมพ์สารบัญเอง \*\*\*

หากจะพิมพ์เนื้อหาที่มีหัวข้อย่อย ให้ใช้ Heading 1,2,3 ของ Word (set ไว้ให้แล้ว)  
เมื่อพิมพ์เสร็จ ให้ไปที่ References => Update Table สารบัญจะอัปเดตให้เอง  
ลองเล่นกับ format ดูก่อนได้ ทำเสร็จแล้วลบกล่องข้อความนี้ทิ้ง

## สารบัญ

<b>บทที่ 1 บทนำ</b>	<b>3</b>
<b>1.1 ที่มา ความสำคัญ</b>	<b>3</b>
<b>1.2 ประโยคปัญหางานวิจัย (Problem Statement)</b>	<b>3</b>
<b>1.3 ผลผลิตและผลลัพธ์ (Outputs and Outcomes)</b>	<b>4</b>
ผลผลิต	4
ผลลัพธ์	4
<b>1.4 ความต้องการของระบบ (Requirements)</b>	<b>4</b>
<b>1.5 ขอบเขตของงานวิจัย (Scopes)</b>	<b>4</b>
<b>1.6 ข้อกำหนดของงานวิจัย (Assumptions)</b>	<b>5</b>
<b>1.7 ขั้นตอนการดำเนินงาน</b>	<b>5</b>
<b>บทที่ 2 ทฤษฎี/งานวิจัย/การศึกษาที่เกี่ยวข้อง</b>	<b>7</b>
<b>2.1[หัวข้อ]</b>	<b>7</b>
2.1.1 [หัวข้อย่อย]	7
<b>2.2[หัวข้อ]</b>	<b>10</b>
<b>บทที่ 3 ระเบียบวิธีวิจัย</b>	<b>14</b>
<b>3.1[หัวข้อ]</b>	<b>14</b>
3.1.1 [หัวข้อย่อย]	15
<b>3.2[หัวข้อ]</b>	<b>15</b>
<b>บทที่ 4 การทดลองและผลการทดลอง/วิจัย</b>	<b>32</b>
<b>4.1[หัวข้อ]</b>	<b>32</b>
4.1.1 [หัวข้อย่อย]	32
<b>4.2[หัวข้อ]</b>	<b>42</b>
<b>บทที่ 5 บทสรุป</b>	<b>43</b>
<b>5.1[หัวข้อ]</b>	<b>43</b>
5.1.1 [หัวข้อย่อย]	43
<b>5.2[หัวข้อ]</b>	<b>43</b>
<b>เอกสารอ้างอิง</b>	<b>44</b>

# บทที่ 1 บทนำ

## 1.1 ที่มา ความสำคัญ

ในปัจจุบันเกมแนว First-Person Shooter (FPS) เป็นประเภทเกมที่ได้รับความนิยมสูง ผู้เล่นส่วนมากให้ความสำคัญกับการเล็งเป้าที่รวดเร็วและแม่นยำ ซึ่งนำไปสู่การพัฒนาเครื่องมือหรือโปรแกรมต่าง ๆ ในการฝึกฝนทักษะการเล็ง เช่น Aim Training Software เป็นต้น อย่างไรก็ตาม กระบวนการฝึกเล็งดังกล่าวยังต้องอาศัยการขยับเมาส์ของผู้เล่นเองเป็นหลัก ซึ่งอาจเกิดข้อจำกัดด้านความแม่นยำและความสม่ำเสมอในการเคลื่อนที่ของมือมนุษย์

โครงการนี้จึงมีแนวคิดในการพัฒนาระบบ "Physical Aimbot" โดยนำหลักการทางด้านวิศวกรรมหุ่นยนต์ (Robotics) และการประมวลผลภาพ (Computer Vision) มาผสมผสานกัน เพื่อสร้างหุ่นยนต์ที่สามารถควบคุมเมาส์จริงบนพื้นผิวจริงได้อย่างแม่นยำและอัตโนมัติ ผ่านการตรวจจับตำแหน่งเป้าหมายบนหน้าจอจากภาพหรือวิดีโอที่ประมวลผลด้วยโมเดล Machine Learning ประเภท Object Detection เช่น YOLO (You Only Look Once) โดยมีวัตถุประสงค์หลักคือ เพื่อศึกษาความเป็นไปได้ในการผสมผสานเทคโนโลยีหุ่นยนต์และ Computer Vision รวมทั้งสร้างความรู้พื้นฐานและประสบการณ์ในการออกแบบและพัฒนาระบบที่มีทั้งองค์ประกอบทางด้าน Software และ Hardware มากกว่าจะมุ่งเน้นการนำไปใช้เพื่อประโยชน์เชิงแข่งขันหรือเชิงโกงภายในเกม

## 1.2 ประโยคปัญหาทางานวิจัย (Problem Statement)

ในการเล็งเป้าหมายในเกม FPS หรือโปรแกรมฝึกเล็งที่ต้องการความแม่นยำสูง ผู้เล่นมักประสบปัญหาในการควบคุมการเคลื่อนที่ของเมาส์ให้แม่นยำและรวดเร็ว เนื่องจากการปรับมุมและตำแหน่งของเมาส์ซ้ำ ๆ ส่งผลให้เสียเวลาและลดประสิทธิภาพในการเล่น โครงการนี้จึงมุ่งสำรวจและพัฒนาระบบการควบคุมการเคลื่อนที่ของเมาส์ด้วยกลไกหุ่นยนต์และการประมวลผลภาพ (Computer Vision) แบบเรียลไทม์ เพื่อสร้างระบบช่วยเล็งที่สามารถตรวจจับเป้าหมายบนหน้าจอและสั่งให้หุ่นยนต์เคลื่อนที่เมาส์ให้ตรงตามเป้าหมายได้อย่างแม่นยำและอัตโนมัติ ทั้งนี้เพื่อเพิ่มความต่อเนื่องในการเล่นและเป็นการศึกษาการผสมผสานเทคโนโลยี Robotics กับ Computer Vision ให้เกิดเป็นรูปธรรม

## 1.3 ผลผลิตและผลลัพธ์ (Outputs and Outcomes)

### ผลผลิต

1. ต้นแบบหุ่นยนต์ (Robot Prototype) ที่สามารถควบคุมและเคลื่อนที่ของเมาส์จริงบนพื้นผิว เพื่อไปถึงตำแหน่งเป้าหมายที่ระบุจากระบบตรวจจับภาพ
2. โมเดล Machine Learning (YOLO-based Object Detection Model) ที่สามารถตรวจจับและระบุตำแหน่งเป้าหมายจากภาพหรือวิดีโอแบบเรียลไทม์ ในโปรแกรมฝึกเล็ง (Aim Training Software) หรือเกม FPS

### ผลลัพธ์

1. ความรู้และความเข้าใจเชิงลึก เกี่ยวกับการผสมผสานระหว่างเทคโนโลยี Robotics และ Computer Vision ในการประยุกต์ใช้สำหรับการควบคุมอุปกรณ์เชิงกายภาพ
2. แนวทางหรือกรอบการทำงาน (Framework) ในการพัฒนาระบบช่วยเล็งแบบกายภาพ (Physical Aiming Assistant) ที่สามารถนำไปต่อยอดหรือปรับปรุงในการศึกษาหรือวิจัยอื่น ๆ ในอนาคตได้

## 1.4 ความต้องการของระบบ (Requirements)

1. ระบบต้องสามารถจับภาพหน้าจอหรือรับข้อมูลจากโปรแกรมฝึกเล็งใน Kovaak โดยใช้ Python library สำหรับการจับภาพ เพื่อระบุพิกัดเป้าหมาย
2. ระบบหุ่นยนต์ต้องสามารถเคลื่อนที่ได้อย่างมีประสิทธิภาพในการเล็งและปรับตำแหน่งเมาส์ตามข้อมูลที่ได้รับ โดยใช้มอเตอร์แบบ Yellow TT motor ที่ติดกับล้อ Mecanum
3. ในอนาคตจะพัฒนาโมดูลสำหรับการส่งสัญญาณคลิก โดยใช้การสื่อสารผ่าน ESP32 Bluetooth ที่เชื่อมต่อกับ PC เพื่อส่งข้อมูลการคลิก
4. ใช้ซอฟต์แวร์และฮาร์ดแวร์ที่หาได้ทั่วไป เช่น Logitech G Pro X Superlight, ESP32 สำหรับการสื่อสาร Bluetooth, และ Python library สำหรับการจับภาพหน้าจอ (แทนการใช้ Capture Card)

## 1.5 ขอบเขตของงานวิจัย (Scopes)

1. ใช้โปรแกรมฝึกเล็ง (Kovaak) เป็นหลักในการเก็บข้อมูลภาพหรือวิดีโอ เพื่อนำมาประมวลผลด้วยโมเดล Machine Learning ในการระบุตำแหน่งเป้าหมาย
2. มุ่งเน้นการทดสอบประสิทธิภาพของระบบ โดยวัดผลจากคะแนน (Score) ที่ได้จากแต่ละ Scenario ในโปรแกรม Kovaak เป็นหลัก เช่น คะแนนรวม (Total Score), ความแม่นยำ (Accuracy), เปอร์เซ็นต์การยิงโดน (Percent

Hit), จำนวนเป้าหมายที่ยิงโดน (Targets Hit), จำนวนเป้าหมายที่พลาด (Targets Missed), และข้อมูลเชิงสถิติอื่นๆ ที่โปรแกรมมีให้ เพื่อศึกษาความสามารถในการเล็งเป้าของระบบหุ่นยนต์อย่างชัดเจนและวัดผลได้

3. ในขั้นตอนนี้จะยังไม่เน้นเก็บสถิติเรื่องความเร็วในการตอบสนอง (Latency) หรือรายละเอียดด้านเวลาเป็นหลัก

4. โครงการนี้มีจุดประสงค์เพื่อศึกษาและพัฒนาองค์ความรู้ในการประยุกต์ใช้ Robotics และ Computer Vision เท่านั้น ไม่มีเจตนาที่จะนำไปใช้งานเพื่อประโยชน์เชิงแข่งขันจริงหรือเพื่อการโกงเกม

## 1.6 ข้อกำหนดของงานวิจัย (Assumptions)

1. สภาพแวดล้อม (Environment) ที่ใช้ในการทดสอบ เช่น ความละเอียดหน้าจอ (Screen Resolution) และการตั้งค่ากราฟิก (Graphic Settings) ในโปรแกรม Kovaak จะถูกกำหนดให้มีค่าคงที่และไม่เปลี่ยนแปลงตลอดช่วงการทดลอง เพื่อความสม่ำเสมอในการเก็บข้อมูล

2. ผู้วิจัยสามารถปรับค่าความไวของเมาส์ (Mouse Sensitivity) และการตั้งค่าอื่นๆ ที่เกี่ยวข้องได้ตามความเหมาะสม เพื่อให้ตรงกับเงื่อนไขและความต้องการในการทดลองร่วมกับโปรแกรม Kovaak

## 1.7 ขั้นตอนการดำเนินงาน

### 1. เก็บข้อมูลและสร้าง Dataset

1.1. บันทึกการเล่นโปรแกรม Kovaak (Scenarios ต่างๆ เช่น Tracking และ Dynamic Clicking)

1.2. ใช้ OpenCV ดึงภาพจากวิดีโอที่บันทึกไว้ นำไปทำ Label และ Augmentation

### 2. พัฒนาและฝึกโมเดล Machine Learning

2.1. บันทึกการเล่นโปรแกรม Kovaak (Scenarios ต่างๆ เช่น Tracking และ Dynamic Clicking)

2.2. ใช้ OpenCV ดึงภาพจากวิดีโอที่บันทึกไว้ นำไปทำ Label และ Augmentation

### 3. ศึกษาและคัดเลือกฮาร์ดแวร์

3.1. สำนักรวบรวมตัวเลือกและตัดสินใจเลือกใช้ล้อ Mecanum และมอเตอร์ประเภท Yellow TT motor เนื่องจากข้อจำกัดด้านงบประมาณ

3.2. ออกแบบระบบจ่ายไฟโดยใช้แบตเตอรี่ 18650 สองก้อน พร้อมวงจรควบคุมแรงดัน (XL4016E1)

### 4. ออกแบบและเขียนแบบ CAD

4.1. ออกแบบโครงสร้างหุ่นยนต์และส่วนยึดอุปกรณ์ทั้งหมด โดยใช้โปรแกรม CAD เพื่อจัดวางตำแหน่งของมอเตอร์, ล้อ, ESP32 และส่วนประกอบอื่นๆ

### 5. สร้างต้นแบบหุ่นยนต์ (Prototype)

5.1. ประกอบวงจรและอุปกรณ์ต่างๆ ตามแบบ CAD

5.2. ทดลองระบบเบื้องต้นเพื่อทดสอบการเคลื่อนที่ของหุ่นยนต์ (ยังไม่รวมการเชื่อมต่อกับระบบ Software)

## 6. ผสานระบบ (Integration)

6.1. เขียนโปรแกรม Python เพื่อจับภาพหน้าจอจากโปรแกรม Kovaak แล้วส่งไปยังโมเดล YOLOv5 ที่ฝึกไว้

6.2. ส่งค่าตำแหน่งเป้าหมายที่ได้จากการตรวจจับไปยัง ESP32 ผ่าน PySerial เพื่อควบคุมการเคลื่อนที่ของหุ่นยนต์

6.3. ใช้ ESP32 Bluetooth ส่งข้อมูลคลิกกลับไปยัง PC (พัฒนาเพิ่มเติมในอนาคต)

## 7. ทดสอบและปรับแต่งระบบ

7.1. ทดสอบการทำงานจริงโดยวัดผลจากคะแนน (Scores) และความแม่นยำ (Accuracy) จาก Kovaak

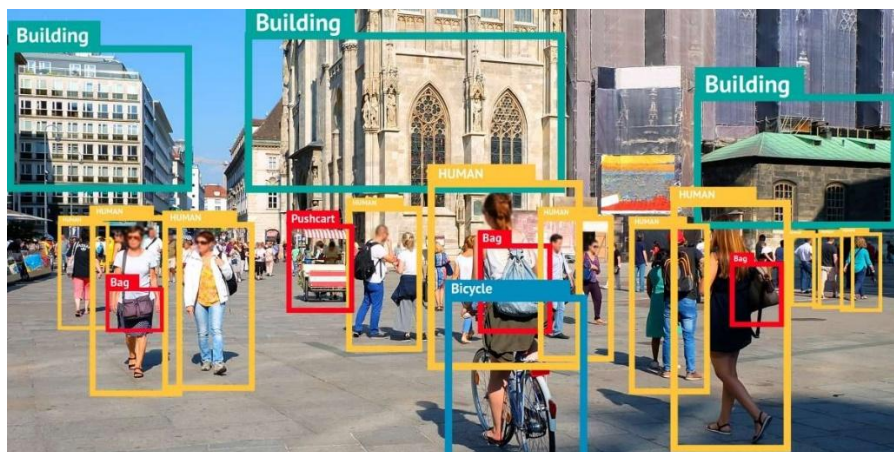
7.2. วิเคราะห์ผลเบื้องต้น และวางแผนเพิ่มประสิทธิภาพ เช่น การปรับปรุง Dataset, การ Train โมเดลเพิ่มเติม, หรือปรับแต่ง Hardware

## บทที่ 2 ทฤษฎี/งานวิจัย/การศึกษาที่เกี่ยวข้อง

[เนื้อหา]

### 2.1[Computer Vision สำหรับ Object Detection]

[การตรวจจับวัตถุ (Object Detection) เป็นเทคนิคในวิทยาการคอมพิวเตอร์ที่ใช้ในการระบุและกำหนดตำแหน่งของวัตถุภายในภาพหรือวิดีโอ โดยทั่วไปจะใช้กรอบสี่เหลี่ยมล้อมรอบวัตถุที่ตรวจพบ]



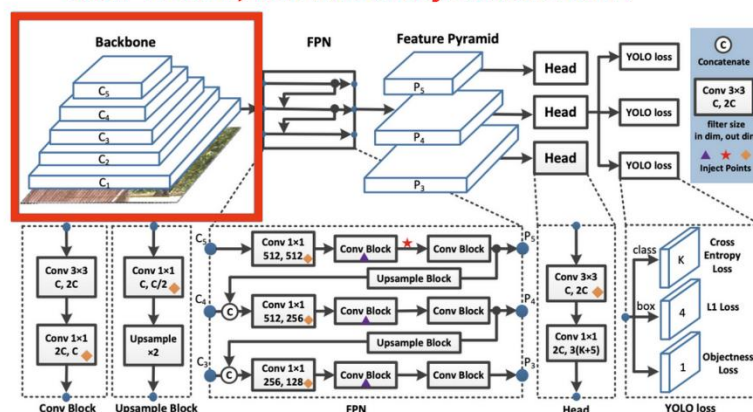
#### 2.1.1 [หลักการดำเนินงานพื้นฐานของ YOLO (You Only Look Once)]

YOLO ทำงานโดยแบ่งภาพอินพุตออกเป็นกริดขนาด  $S \times S$  และสำหรับแต่ละเซลล์ในกริดนั้น โมเดลจะทำนาย  $B$  กล่องขอบเขต (bounding boxes) พร้อมกับคะแนนความเชื่อมั่น (confidence scores) ที่บ่งบอกถึงความน่าจะเป็นที่กล่องนั้นจะมีวัตถุอยู่ และความแม่นยำของกล่องที่ทำนาย นอกจากนี้ แต่ละเซลล์ยังทำนายความน่าจะเป็นแบบมีเงื่อนไขสำหรับแต่ละคลาสของวัตถุ โดยอิงตามความน่าจะเป็นที่เซลล์นั้นมีวัตถุอยู่ การทำนายทั้งหมดนี้เกิดขึ้นพร้อมกันในการประมวลผลเพียงครั้งเดียว ทำให้ YOLO มีความเร็วสูงในการตรวจจับวัตถุ





Until YOLOs, this has always been a CNN



- การแบ่งภาพเป็นกริด: ภาพอินพุตจะถูกแบ่งออกเป็นกริดขนาด  $S \times S$  โดยแต่ละเซลล์ในกริดจะรับผิดชอบในการทำนายวัตถุที่มีศูนย์กลางอยู่ในเซลล์นั้น
- การทำนายกล่องขอบเขตและคะแนนความเชื่อมั่น: สำหรับแต่ละเซลล์ โมเดลจะทำนาย กล่องขอบเขต พร้อมกับคะแนนความเชื่อมั่นที่บ่งบอกถึงความน่าจะเป็นที่กล่องนั้นจะมีวัตถุอยู่ และความแม่นยำของกล่องที่ทำนาย
- การทำนายความน่าจะเป็นของคลาสวัตถุ: โมเดลจะทำนายความน่าจะเป็นแบบมีเงื่อนไขสำหรับแต่ละคลาสของวัตถุ โดยอิงตามความน่าจะเป็นที่เซลล์นั้นมีวัตถุอยู่
- การคูณคะแนนความเชื่อมั่นกับความน่าจะเป็นของคลาส: คะแนนความเชื่อมั่นจะถูกคูณกับความน่าจะเป็นของคลาสเพื่อให้ได้คะแนนสุดท้ายสำหรับแต่ละกล่องขอบเขต
- การกรองกล่องขอบเขต: ใช้เทคนิค **Non-Maximum Suppression (NMS)** เพื่อลบกล่องขอบเขตที่ซ้ำซ้อนและเก็บเฉพาะกล่องที่มีคะแนนสูงสุดสำหรับแต่ละวัตถุ

## 2.1.2 [การเปรียบเทียบรุ่น YOLO (YOLO Version Comparison)]

ในงานนี้ เราเน้นการใช้โมเดล YOLOv5 ซึ่งมีหลายเวอร์ชันที่แตกต่างกันในเรื่องขนาดและประสิทธิภาพ เราสามารถแบ่งเวอร์ชันหลัก ๆ ได้ดังนี้:

### 1. YOLOv5s (Small)

- ข้อดี:
  - มีขนาดเล็ก ใช้ทรัพยากรน้อย

- ประมวลผลได้รวดเร็ว เหมาะกับงานที่ต้องการการตอบสนองแบบเรียลไทม์ เช่น การตรวจจับเป้าหมายในโปรแกรม Aim Training (Kovaak)
- ข้อเสีย:
  - ความแม่นยำอาจต่ำกว่าเมื่อเทียบกับเวอร์ชันที่มีขนาดใหญ่กว่า
- 2. YOLOv5m (Medium)
  - ข้อดี:
    - ให้ความแม่นยำสูงขึ้นจาก YOLOv5s ด้วยจำนวนพารามิเตอร์ที่เพิ่มขึ้น
    - ยังสามารถประมวลผลแบบเรียลไทม์ได้ดีในระบบที่มีฮาร์ดแวร์ประสิทธิภาพสูง
  - ข้อเสีย:
    - ใช้ทรัพยากรมากขึ้น และอาจมีความช้ากว่า YOLOv5s เล็กน้อย
- 3. YOLOv5l (Large) และ YOLOv5x (Extra Large)
  - ข้อดี:
    - ให้ความแม่นยำสูงสุด เนื่องจากมีจำนวนพารามิเตอร์มากที่สุด
  - ข้อเสีย:
    - ประมวลผลช้ากว่าและต้องการทรัพยากรระบบที่สูงขึ้น ซึ่งอาจไม่เหมาะกับงานที่ต้องการความเร็วแบบเรียลไทม์ในสภาพแวดล้อมที่มีข้อจำกัดด้านทรัพยากร

สำหรับโครงการนี้ เราเริ่มต้นด้วย YOLOv5s (Small) เนื่องจากมีความเร็วสูงและเหมาะสมกับการทดลองเบื้องต้นในระบบ Aim Training (Kovaak) อย่างไรก็ตาม ด้วยประสิทธิภาพของ PC ที่มีอยู่ (สามารถรันได้ 200+ FPS) ในอนาคตจึงมีความเป็นไปได้ที่จะปรับปรุงและฝึกเพิ่มเติมโดยพิจารณาให้เหมาะสมกับข้อจำกัดของฮาร์ดแวร์และลักษณะของงาน เช่น หากต้องการตรวจจับอย่างรวดเร็วและใช้งานร่วมกับระบบที่มีข้อจำกัดด้าน latency อาจเลือกใช้ YOLOv5s หรือ m ในขณะที่หากระบบสามารถรองรับการประมวลผลที่ช้าลงได้และต้องการความแม่นยำสูง YOLOv5l หรือ x ก็เป็นทางเลือกที่เหมาะสม

## 2.2[Robotics และการควบคุมการเคลื่อนที่]

[เนื้อหา]

### 2.2.1 [Omni Wheels และ Mecanum Wheels]

ในการออกแบบระบบเคลื่อนที่สำหรับหุ่นยนต์ที่ควบคุมเมคานิกส์ในงาน Physical Aimbot มีล้อสองแบบที่มักถูกนำมาใช้ได้แก่ Omni Wheels และ Mecanum Wheels:

- Omni Wheels:

- หลักการ: ล้อ Omni มีแผ่นสัมผัสที่สามารถหมุนได้ในแนวตั้งและมีล้อเล็กติดอยู่รอบๆ ซึ่งช่วยให้เคลื่อนที่ได้ในทิศทางหลายทิศทาง
- ข้อดี: ให้ความสามารถในการเคลื่อนที่แบบ holonomic (เคลื่อนที่ในทุกทิศทาง)
- ข้อเสีย: โดยทั่วไปราคาแพงกว่าแบบอื่นถึง 4–5 เท่า ทำให้ไม่เหมาะสำหรับงานที่มีข้อจำกัดด้านงบประมาณ
- Mecanum Wheels:
  - หลักการ: ล้อ Mecanum มีลูกกลิ้งติดอยู่ที่ขอบล้อในมุมเอียง ช่วยให้สามารถเคลื่อนที่แบบ omnidirectional ได้ด้วยการควบคุมความเร็วของแต่ละล้อแยกกัน
  - ข้อดี: ราคาถูกกว่า Omni Wheels อย่างมาก, ให้ความสามารถในการเคลื่อนที่แบบหลายทิศทางได้ดีเพียงพอสำหรับงานควบคุมเมาส์ในระบบ Physical Aimbot
  - ข้อเสีย: การควบคุมอาจซับซ้อนกว่าเล็กน้อยเนื่องจากต้องคำนวณการเคลื่อนที่ที่สัมพันธ์กันของล้อทั้งหมด

สาเหตุที่เลือก Mecanum Wheels:

เนื่องจากข้อจำกัดด้านงบประมาณ โดย Omni Wheels มีราคาสูงถึง 4–5 เท่าของ Mecanum Wheels ทำให้การเลือกใช้ Mecanum Wheels เป็นทางเลือกที่ประหยัดและยังคงให้ประสิทธิภาพในการเคลื่อนที่ที่เพียงพอสำหรับการควบคุมเมาส์ในโครงการนี้

### 2.2.2 [ทฤษฎี Inverse Kinematics สำหรับล้อ Mecanum]

การควบคุมการเคลื่อนที่ในหุ่นยนต์ที่ใช้ล้อ Mecanum ต้องอาศัยหลักการของ Inverse Kinematics ซึ่งเป็นการคำนวณหาความเร็วและทิศทางของแต่ละล้อจากคำสั่งการเคลื่อนที่ที่ต้องการ (เช่น การเล็งเป้าหมาย)

- **หลักการคำนวณ:**

สำหรับหุ่นยนต์ที่ใช้ล้อ Mecanum จำนวน 4 ตัว เราสามารถคำนวณความเร็วของแต่ละล้อจากความต้องการเคลื่อนที่ในแกน X ( $V_x$ ), แกน Y ( $V_y$ ) และการหมุน (angular velocity,  $\omega$ ) ได้ดังนี้:

- ความเร็วล้อหน้าซ้าย ( $V_{FL}$ ) =  $V_x - V_y - (L + W) * \omega$
- ความเร็วล้อหน้าขวา ( $V_{FR}$ ) =  $V_x + V_y + (L + W) * \omega$
- ความเร็วล้อหลังซ้าย ( $V_{RL}$ ) =  $V_x + V_y - (L + W) * \omega$
- ความเร็วล้อหลังขวา ( $V_{RR}$ ) =  $V_x - V_y + (L + W) * \omega$

โดยที่:

- **$V_x$ :** ความเร็วในแนวแกน X
- **$V_y$ :** ความเร็วในแนวแกน Y
- **$\omega$ :** ความเร็วเชิงมุม (การหมุน)

- **L:** ระยะจากจุดศูนย์กลางไปยังล้อในแนวตั้ง
- **W:** ระยะจากจุดศูนย์กลางไปยังล้อในแนวนอน
- **แนวคิดเบื้องต้นในการควบคุม:**  
เมื่อระบบ Computer Vision ตรวจจับเป้าหมายแล้ว ระบบจะคำนวณตำแหน่งเป้าหมายที่ต้องเล็ง จากนั้นใช้ Inverse Kinematics ในการแปลงคำสั่งเคลื่อนที่ (เช่น ค่า  $V_x$ ,  $V_y$ ,  $\omega$ ) ไปเป็นความเร็วที่แต่ละล้อควรหมุนเพื่อให้หุ่นยนต์เคลื่อนที่ไปในทิศทางที่ต้องการได้อย่างแม่นยำ การคำนวณเหล่านี้ช่วยให้ระบบสามารถปรับตำแหน่งเมาส์ให้ตรงกับเป้าหมายได้โดยอัตโนมัติ

## 2.3[Hardware และการสื่อสารข้อมูล]

[เนื้อหา]

### 2.3.1 [ESP32 Microcontroller]

เราเลือกใช้ ESP32 เนื่องจากมีคุณสมบัติที่โดดเด่นหลายประการที่สอดคล้องกับความต้องการของโครงการนี้

- **ความเร็วและประสิทธิภาพสูง:** ESP32 สามารถประมวลผลคำสั่งและสื่อสารได้อย่างรวดเร็ว ทำให้รองรับงานที่ต้องการการตอบสนองแบบเรียลไทม์
- **การเชื่อมต่อแบบ WiFi และ Bluetooth:** การมีทั้ง WiFi และ Bluetooth ในตัวช่วยให้เราสามารถเชื่อมต่อกับ PC ผ่าน WiFi หรือใช้ Bluetooth สำหรับส่งคำสั่งการคลิกโดยตรง ซึ่งจะช่วยลดความซับซ้อนของโมดูลคลิกแยกต่างหาก
- **ความยืดหยุ่นในการประมวลผล Inverse Kinematics:** ระบบสามารถเลือกที่จะคำนวณ Inverse Kinematics บนตัว ESP32 เอง หรือรับค่าที่คำนวณจาก PC (เช่น ค่า Distance, Degree และ Shooting command) ขึ้นอยู่กับประสิทธิภาพของแต่ละหน่วย เราสามารถปรับเปลี่ยนวิธีการส่งข้อมูลได้โดยพิจารณาจากความเร็วในการคำนวณของ PC เทียบกับ ESP32

### 2.3.2 [การสื่อสารข้อมูลผ่าน PySerial]

สำหรับการส่งข้อมูลระหว่างคอมพิวเตอร์และ ESP32 เราใช้การสื่อสารผ่าน PySerial ด้วยข้อดีดังนี้:

- **การเชื่อมต่อที่ง่าย:** PySerial ช่วยให้การส่งข้อมูลระหว่าง PC กับ ESP32 เป็นไปอย่างราบรื่น โดยไม่ต้องใช้ฮาร์ดแวร์เพิ่มเติม
- **ประเภทของข้อมูลที่ส่ง:**
  - **Distance:** ระยะห่างระหว่างตำแหน่งปัจจุบันกับเป้าหมายที่คำนวณได้
  - **Degree:** มุมที่เป้าหมายอยู่ในระบบพิกัด
  - **Shooting Command:** คำสั่งยิงที่จะส่งไปยัง ESP32 เมื่อเป้าหมายอยู่ในขอบเขตที่กำหนด

- กระบวนการส่งข้อมูล:
  - หากการคำนวณ Inverse Kinematics ถูกดำเนินการบน PC เราจะส่งข้อมูลที่คำนวณแล้ว (Distance, Degree) พร้อมกับคำสั่งยิง (Shooting command) ไปยัง ESP32
  - ในทางกลับกัน หาก ESP32 มีความสามารถในการคำนวณ Inverse Kinematics ได้เร็วเพียงพอ ระบบจะรับข้อมูลพื้นฐานจาก PC (เช่น ตำแหน่งเป้าหมาย) แล้วคำนวณค่า Distance, Degree และตัดสินใจส่งคำสั่งยิงภายในตัวเอง

### 2.3.3 [Motor และ Driver]

ในการขับเคลื่อนระบบหุ่นยนต์ เราเลือกใช้ **Yellow TT Motor** ร่วมกับ **DRV8833 Motor Driver** ด้วยเหตุผลดังนี้:

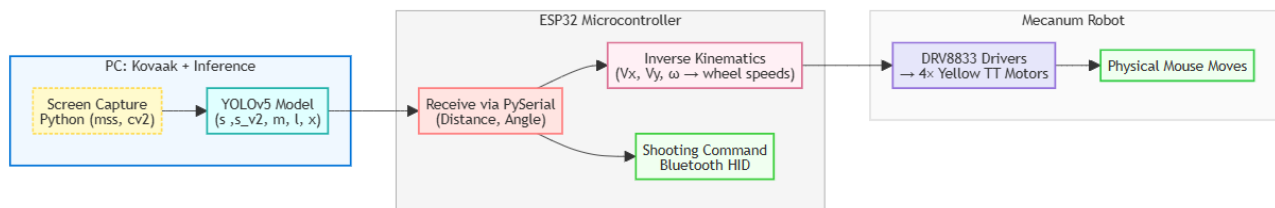
- **Yellow TT Motor:**
  - ร้านค้าที่มีจำหน่ายมีให้เลือกเพียง 2 แบบคือ 1:150 และ 1:48 สำหรับเกียร์
  - เนื่องจากหุ่นยนต์ของเราเบา (ใช้สำหรับเคลื่อนที่เมสที่มีน้ำหนักต่ำ) เราจึงเลือกใช้เกียร์ 1:48 เพื่อให้ได้ความเร็วที่สูงขึ้น แม้ว่าแรงบิด (torque) จะน้อยลง แต่สำหรับงานนี้ ความเร็วในการตอบสนองและการเคลื่อนที่เป็นสิ่งที่สำคัญกว่า
- **DRV8833 Motor Driver:**
  - เป็นตัวควบคุมมอเตอร์ที่มีขนาดกะทัดรัด น้ำหนักเบา และรองรับการทำงานของมอเตอร์ในระดับแรงดันที่เหมาะสมสำหรับระบบของเรา

## บทที่ 3 ระเบียบวิธีวิจัย

[เนื้อหา]

### 3.1[ภาพรวมของระบบ (System Overview)]

[โครงสร้างภาพรวมของระบบนี้แบ่งออกเป็นหลายโมดูลที่ทำงานร่วมกันเพื่อให้ได้ระบบ Physical Aimbot ที่สามารถจับภาพจาก Kovaak, ประมวลผลข้อมูลด้วยโมเดล YOLO, คำนวณ Inverse Kinematics และควบคุมการเคลื่อนที่ของหุ่นยนต์ พร้อมทั้งส่งคำสั่งคลิกกลับไปยัง PC ผ่าน ESP32]



ส่วนประกอบหลักของระบบมีดังนี้:

#### 1. Screen Capture Module:

- โปรแกรม Python ใช้จับภาพหน้าจอจากโปรแกรม Kovaak
- ภาพที่จับได้ถูกส่งต่อไปยังโมเดล YOLO เพื่อทำการตรวจจับวัตถุ

#### 2. Object Detection Module (YOLO Model):

- โมเดล YOLO (เช่น YOLOv5s หรือ YOLOv5m ตามที่เลือก) ประมวลผลภาพที่ได้รับจาก Screen Capture
- ตรวจจับตำแหน่งเป้าหมาย (bounding boxes, confidence scores) และส่งข้อมูลผลลัพธ์ไปยัง ESP32

#### 3. ESP32 Microcontroller:

- รับข้อมูลจากโมเดล YOLO ผ่านการสื่อสาร (PySerial)
- คำนวณ Inverse Kinematics (หรือรับค่าที่คำนวณจาก PC ขึ้นอยู่กับประสิทธิภาพ) เพื่อแปลงข้อมูลตำแหน่งเป้าหมายเป็นคำสั่งสำหรับควบคุมมอเตอร์ (เช่น ค่าความเร็วและมุมของแต่ละล้อ)
- สั่งการเคลื่อนที่ของหุ่นยนต์ตามข้อมูลที่คำนวณได้
- ส่งคำสั่งการคลิก (Shooting Command) ผ่าน Bluetooth กลับไปยัง PC เมื่อเป้าหมายอยู่ในขอบเขตที่กำหนด

#### 4. Motor Control Module:

- มอเตอร์ (Yellow TT Motor) ที่ติดตั้งกับล้อ Mecanum รับคำสั่งจาก ESP32 เพื่อขับเคลื่อนหุ่นยนต์ให้เคลื่อนที่ไปตามตำแหน่งที่ต้องการ

## 5. Communication Flow:

- **PC:** จับภาพหน้าจอจาก Kovaak → ส่งข้อมูลไปยังโมเดล YOLO
- **Model:** ตรวจจับเป้าหมาย → ส่งผลลัพธ์ (ตำแหน่งเป้าหมาย) ไปยัง ESP32
- **ESP32:** คำนวณ Inverse Kinematics → สั่งมอเตอร์เคลื่อนที่ → ส่งคำสั่งคลิกกลับไปยัง PC

### 3.1.1 [หัวข้อย่อย]

1. เนื้อหา
2. เนื้อหา

## 3.2[วิธีการบันทึกวิดีโอการเล่นโปรแกรม Kovaak]

[เนื้อหา]

### 3.2.1 [วิธีการบันทึกวิดีโอการเล่นโปรแกรม Kovaak]

ในการสร้าง Dataset สำหรับตรวจจับเป้าหมาย (Targets) ในโปรแกรม Aim Training อย่าง Kovaak ผู้วิจัยได้ทำการบันทึกวิดีโอของการเล่นทั้งหมด 5 ฉาก (Scenarios) ซึ่งประกอบด้วย:

1. **3D Switching Robots**
2. **Buff Robots 360**
3. **Close Fast Strafes Easy Invincible Robot 25% Slower**
4. **CLS Click Robots Rounded**
5. **Correction Accuracy – Close I Strafe Robot**

- **ระยะเวลา:** แต่ละฉากมีความยาวประมาณ 1 นาที รวมเป็นเวลาทั้งสิ้นประมาณ 5 นาที
- **ความละเอียด (Resolution):** 1920 × 1080
- **อัตราเฟรม (Frame Rate):** 30 FPS

โดยใช้โปรแกรมบันทึกวิดีโอหน้าจอ (Screen Recording) ในสภาพแวดล้อมการเล่นปกติของ Kovaak เพื่อให้ได้ข้อมูลการเล็งเป้าหมายที่ใกล้เคียงการใช้งานจริงที่สุด

### 3.2.1 [การดึงภาพจากวิดีโอด้วย OpenCV และขั้นตอนการ Label ข้อมูลเป้าหมาย (Object Labeling)]

หลังจากได้ไฟล์วิดีโอทั้ง 5 ชิ้นแล้ว ผู้วิจัยจึงนำวิดีโอเหล่านั้นมาประมวลผลด้วย **Python** และ **OpenCV** เพื่อแยกเฟรมออกมาเป็นรูปภาพ (.png) ดังตัวอย่างโค้ดด้านล่าง:

```
import cv2
import os

# Define the path for the videos and the output directory
video_path = r"D:\UNIVERSITY\YR3\FRA361_Open_Topic\DATASET\VIDEO"
output_base_path = r"D:\UNIVERSITY\YR3\FRA361_Open_Topic\DATASET\PICTURE"

# List of video files in the folder
video_files = [
    "3D Switching Robots.mp4",
    "Buff Robots 360.mp4",
    "Close Fast Strafes Easy Invincible Robot 25% Slower.mp4",
    "CLS Click Robots Rounded.mp4",
    "Correction Accuracy - Close I Strafe Robot.mp4"
]

# Iterate through each video file
for video_file in video_files:
    video_full_path = os.path.join(video_path, video_file)
    cap = cv2.VideoCapture(video_full_path)

    if not cap.isOpened():
        print(f"Error: Could not open video file {video_file}")
        continue

    # Create a folder to store frames
    output_folder = os.path.join(output_base_path, video_file.split('.')[0])
    os.makedirs(output_folder, exist_ok=True)

    frame_count = 0
    success, frame = cap.read()

    while success:
        frame_filename = os.path.join(output_folder, f"frame_{frame_count:04d}.png")
        cv2.imwrite(frame_filename, frame)

        success, frame = cap.read()
        frame_count += 1

    cap.release()
    print(f"Frames extracted for {video_file} -> {output_folder}")

print("All videos processed.")
```

#### 1. กระบวนการดึงภาพ (Frame Extraction):

- โปรแกรมจะเปิดวิดีโอแต่ละไฟล์และอ่านเฟรมทีละเฟรม
- แต่ละเฟรมจะถูกบันทึกเป็นไฟล์ภาพ .png ไว้ในโฟลเดอร์แยกตามชื่อวิดีโอ



- ทำให้ได้จำนวนภาพทั้งหมดหลายพันภาพ (ขึ้นอยู่กับความยาววิดีโอและอัตราเฟรม)

## 2. การ Label ข้อมูล (Object Labeling):

- นำภาพที่ได้ไปทำ **Label** จุด (Bounding Box) ของ “Robot” ซึ่งเป็นเป้าหมายหลักในการฝึก
- รูปแบบการบันทึก Label เป็น **YOLO v5 PyTorch format** เพื่อให้สอดคล้องกับโมเดล YOLO ที่จะใช้งานต่อไป
- ใช้เครื่องมือออนไลน์ (เช่น Roboflow หรือโปรแกรม Labeling อื่น ๆ) เพื่อช่วยในการ Annotate วัตถุ

### 3.2.3 [การทำ Augmentation ผ่าน Roboflow]

หลังจากได้ภาพที่ Label เสร็จเรียบร้อยแล้ว ผู้วิจัยได้ทำ **Data Augmentation** โดยใช้แพลตฟอร์ม **Roboflow** เพื่อเพิ่มความหลากหลายของภาพและลดปัญหา **Overfitting** ซึ่งรายละเอียดของการ Augment มีดังนี้:

#### 1. จำนวน Dataset:

- Dataset มีภาพทั้งหมดประมาณ **11,342** ภาพ (รวมภาพต้นฉบับและภาพที่ผ่านการ Augmentation)

#### 2. การปรับขนาด (Resize):

- แต่ละภาพถูกปรับขนาดเป็น **640 × 640** (Stretch) เพื่อให้ตรงกับ input size ของ YOLOv5

#### 3. การ Flip และ Crop:

- **Horizontal Flip** (50% Probability) ช่วยจำลองตำแหน่งของศัตรูที่อาจโผล่มาทั้งด้านซ้ายและขวาในฉากเกม เพื่อให้โมเดลไม่ Bias ด้านใดด้านหนึ่ง
- **Vertical Flip** (50% Probability) ช่วยจำลองตำแหน่งของศัตรูที่อาจโผล่มาทั้งด้านซ้ายและขวาในฉากเกม เพื่อให้โมเดลไม่ Bias ด้านใดด้านหนึ่ง
- **Random Crop** ระหว่าง 0% ถึง 20% ของ Bounding Box ช่วยให้โมเดลเรียนรู้จากสถานการณ์ที่เป้าหมายบางส่วนอาจถูกซ่อนอยู่หลังวัตถุหรือขอบจอ เช่น เป้าหมายหลบอยู่หลังของฉากในเกม

#### 4. การ Shear (บิดภาพ):

- มีการสุ่มค่าการ Shear ในช่วง **-10° ถึง +10°** ทั้งแนวนอนและแนวตั้ง ใช้เพื่อจำลองความเบลอหรือ distortion จากการเคลื่อนที่เร็วในเกม (เช่น ตอนเป้าหมาย **strafe** อย่างรวดเร็ว)

#### 5. Auto-Orientation และ EXIF Stripping:

- มีการปรับ Orientation ของภาพอัตโนมัติเพื่อให้ทิศทางของภาพถูกต้องเสมอ
- ลบข้อมูล EXIF ที่ไม่จำเป็นออก

การ **Augmentation** ดังกล่าวช่วยเพิ่มความหลากหลายของภาพและสถานการณ์การเดินเท้า ทำให้โมเดลเรียนรู้ได้ครอบคลุมมากขึ้น อย่างไรก็ตาม นี่เป็นการตั้งค่าพื้นฐานในการ **Augment** ครั้งแรกเท่านั้น ซึ่งวางแผนที่จะปรับปรุงหรือเปลี่ยนแปลงพารามิเตอร์ในอนาคต เมื่อมีการปรับแต่งระบบหุ่นยนต์และโมเดล YOLO ให้มีประสิทธิภาพมากขึ้น

### 3.2.4 [การสร้าง **Dataset v2** (Scenarios ขยาย + Quality Control)]

เนื่องจาก **Dataset** ชุดแรก มีปริมาณภาพค่อนข้างน้อย และครอบคลุมเพียง 5 Scenarios ที่มีลักษณะคล้ายกัน เช่น robot เดินตรงใน map เดิม ทำให้โมเดลที่ฝึกจากข้อมูลดังกล่าวมีแนวโน้ม **overfit** และ ขาดความสามารถในการ **generalize** เมื่อนำไปใช้กับฉากใหม่ ๆ ที่มีลักษณะแตกต่างกัน เช่น robot ซ้อนกันหลายตัว / แสงต่างกัน / map คนละแบบ

นอกจากนี้ในเวอร์ชันแรกมีการ label อย่างเร่งรีบ จึงอาจเกิด **miss-label** และ **noise** ในชุดข้อมูล ได้สูง ผู้วิจัยจึงตัดสินใจสร้าง **Dataset v2** โดยปรับปรุงทั้งด้าน ปริมาณ, คุณภาพ, และ ความหลากหลายของสถานการณ์

**Scenarios ใหม่ที่เพิ่ม:**

เพิ่ม 10 ฉากใหม่ที่ครอบคลุมลักษณะการเล่นที่หลากหลาย เช่น **tracking, click-timing, low-ground jumps, long-range flicks** เพื่อให้ **Dataset** มีความหลากหลายและครอบคลุมสถานการณ์จริงมากขึ้น

**การดิงเฟรม:**

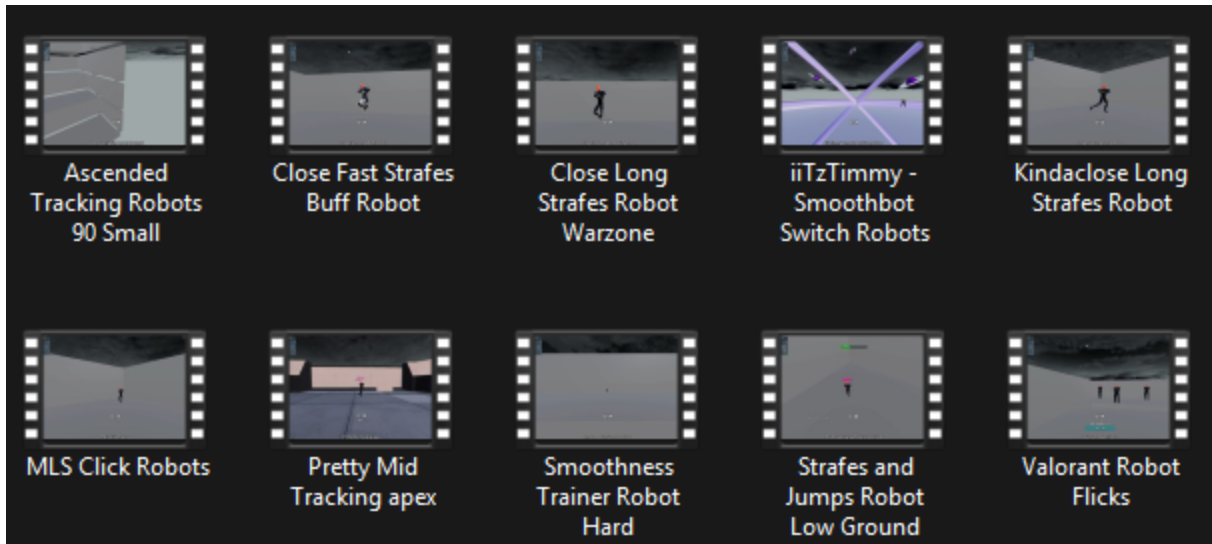
ลด **sampling** เป็น 5 fps (เลือกทุก 6 เฟรมจากวิดีโอ 30 fps ความยาว 1 นาที) เพื่อหลีกเลี่ยงข้อมูลซ้ำซ้อนเกินไป และยังช่วยให้สามารถเพิ่มจำนวน **scenario** ได้มากขึ้นภายใต้จำนวนเท่าเดิม

**จำนวนภาพหลัง extraction:**

ได้ภาพทั้งหมด 38,409 ภาพ ซึ่งมากกว่าชุดก่อนหน้าถึง 3.4 เท่า

**การตรวจสอบ label:**

มีการตรวจสอบ **bounding box** เดิมอย่างละเอียด และแก้ไข **miss-label** ที่พบก่อนจะทำการ **export** เป็น YOLO v5 (PyTorch format) เพื่อเตรียมใช้ในการฝึกโมเดลจริง



ภาพ Dataset ชุดใหม่ที่เพิ่มขึ้น

#### 3.2.4.1 Pre-processing

- **Auto-orientation & EXIF strip** – ป้องกันปัญหาการกลับหัวเวลาภาพมาจากเครื่องต่างแพลตฟอร์ม
- **Adaptive equalization** – เพิ่ม contrast ใน map ที่มี *fog* หรือ *low-light* (เช่นจาก Valorant Robot Flicks)
- **Resize 640 × 640 (Stretch)** – ตรงตามค่า --img 640 ของโมเดลทุกขนาด

#### 3.2.4.2 Augmentation (7× per source)

มีการใช้เทคนิค Augmentation หลากหลายรูปแบบเพื่อจำลองสถานการณ์ต่าง ๆ ที่เกิดขึ้นจริงในเกม โดยมีรายละเอียดดังนี้:

- **Horizontal Flip (50%)**  
เพื่อจำลองการเคลื่อนไหวของเป้าหมายที่ *strafe* ซ้าย/ขวาสลับไปมาอย่างต่อเนื่องในเกม FPS ซึ่งเป็นหนึ่งในสถานการณ์ที่พบได้บ่อย
- **Random Rotation ( $\pm 15^\circ$ )**  
ใช้เพื่อเลียนแบบการส่ายกล้องเร็ว หรือการเคลื่อนไหวแบบ flick ของผู้เล่นที่หมุนเล็งเป้าแบบฉับพลัน
- **Brightness Adjustment ( $-20\%$  ถึง  $+20\%$ ) และ Exposure ( $\pm 10\%$ )**  
จำลองการเปลี่ยนแปลงของแสงใน Arena หรือแผนที่ ซึ่งเกิดขึ้นได้เมื่อกำลังหมุนผ่านจุดที่มีความสว่างต่างกัน เช่น โชนแสงแดดและโชนเงา
- **Box Crop (0–20%)**  
เพื่อจำลองสถานการณ์ที่เป้าหมายโผล่แค่ครึ่งตัว เช่น เมื่อยืนหลังกำแพง สิ่งกีดขวาง หรือโผล่จากมุมบัง
- **Shear ( $\pm 10^\circ$ ), Gaussian Blur (0–0.1px), และ Salt-Pepper Noise (0.06%)**  
ใช้จำลอง motion blur และ noise ที่เกิดขึ้นจริงเมื่อมีการเคลื่อนไหวเร็ว เช่น การวิ่งหรือเป้าหมายหมุนตัว

### 3.3[การฝึกและประเมินผลโมเดล YOLO (Model Training and Evaluation)]

[เนื้อหา]

#### 3.3.1 [สภาพแวดล้อมที่ใช้ในการฝึกโมเดล (Hardware Spec, Software Environment, Libraries)]

- ฮาร์ดแวร์ (Hardware):
  - ใช้เครื่อง ASUS TUF A15 Laptop ที่มาพร้อมกับ RTX 3060
  - CPU: Ryzen 7 6700H
  - RAM: 16GB DDR5 (ความถี่ 4800Hz)
  - SSD สำหรับความเร็วในการอ่านเขียนข้อมูล
- สภาพแวดล้อมการฝึก (Training Environment):
  - ใช้ Conda Environment สำหรับการติดตั้ง Python libraries ที่เกี่ยวข้องกับ YOLOv5
  - Libraries ที่สำคัญ: PyTorch, OpenCV, NumPy, และอื่น ๆ ตามที่ระบุใน requirements.txt ของ YOLOv5

##### 3.3.1.1 [การย้ายและปรับปรุงสภาพแวดล้อมการฝึกโมเดล]

หลังจากการฝึกโมเดลรอบแรกเสร็จสิ้น ผู้วิจัยพบว่าเครื่องคอมพิวเตอร์ส่วนตัวที่ใช้ในการฝึก (ASUS TUF A15, GPU: RTX 3060) เริ่มมีข้อจำกัดด้านเวลาและประสิทธิภาพเมื่อใช้กับ Dataset ที่มีขนาดใหญ่ขึ้น (Dataset v2)

เมื่อเริ่มทดลองเทรนโมเดลขนาดกลาง (YOLOv5m) พบปัญหาหลายประการ ได้แก่:

- ระยะเวลาในการฝึก 1 epoch ใช้นานมาก (ประมาณ 60 นาทีต่อ epoch)
- เทรนยังไม่ทันครบรอบเกิด error ด้านหน่วยความจำ เช่น out of memory / CUDA crash
- ไม่สามารถเพิ่ม batch size ได้เกิน 12 โดยไม่เกิดปัญหา

ด้วยเหตุนี้ ผู้วิจัยจึงตัดสินใจ ย้ายการฝึกทั้งหมดไปที่เซิร์ฟเวอร์ของคณะ ที่มีทรัพยากรสูงกว่าอย่างชัดเจน โดยมีสเปคดังนี้:

การใช้งาน GPU และระบบ Remote:

ระบบใช้ GPU แบบ Dual Tesla V100 (หน่วยละ 32 GB VRAM) โดยใช้งานผ่านการ เชื่อมต่อระยะไกล (Remote) ด้วย SSH ทั้งหมด เพื่อประมวผลและฝึกโมเดล

สามารถฝึกโมเดลได้พร้อมกัน 2 โมเดล โดยแยกการใช้งาน GPU ออกจากกันชัดเจน

- โมเดลที่ 1 ใช้งานด้วยคำสั่ง --device 0

- โมเดลที่ 2 ใช้งานด้วยคำสั่ง `--device 1`

เพื่อหลีกเลี่ยงการชนกันของ **Cuda core** และใช้ทรัพยากรอย่างมีประสิทธิภาพสูงสุด

ทำให้สามารถเพิ่ม **batch size** ได้มากขึ้น, ใช้เวลาน้อยลงกว่าเดิมหลายเท่า และลดความเสี่ยงเรื่องแรมไม่จบหรือเครื่องค้าง

### 3.3.2 [ขั้นตอนการ Train โมเดล YOLOv5 (เวอร์ชัน Small) และการตั้งค่าพารามิเตอร์ (Hyperparameter Tuning)]

#### 3.3.2.1 การฝึก YOLOv5s รุ่นแรก (Training V1 – Dataset v1)

- การเตรียมข้อมูล:
  - ใช้ Dataset ที่ได้จากขั้นตอน 3.2 ซึ่งประกอบด้วยภาพที่ Label แล้วในรูปแบบ YOLO v5 PyTorch format

- คำสั่งการฝึก (Training Command):

ใช้คำสั่งต่อไปนี้เพื่อฝึกโมเดล YOLOv5s:

```
python train.py --img 640 --batch 16 --epochs 100 --data
"D:/UNIVERSITY/YR3/FRA361_Open_Topic/DATASET/For
Train/FRA361_OpenTopic_Aimbot.v1i.yolov5pytorch/data.yaml" --cfg models/yolov5s.yaml
--weights yolov5s.pt --device 0 --optimizer Adam --noautoanchor --sync-bn
```

- อธิบาย **Parameter**:

- `--img 640`: ปรับขนาดภาพเป็น 640×640
- `--batch 16`: ใช้ batch size 16
- `--epochs 100`: ฝึกโมเดลเป็นเวลา 100 epochs
- `--data .../data.yaml`: ระบุมุมุมูล Dataset
- `--cfg models/yolov5s.yaml`: กำหนด configuration ของ YOLOv5s
- `--weights yolov5s.pt`: เริ่มต้นด้วย pre-trained weights ของ YOLOv5s
- `--device 0`: ใช้ GPU ที่มี ID 0
- `--optimizer Adam`: ใช้optimizer Adam
- `--noautoanchor`: ปิดการสร้าง anchor โดยอัตโนมัติ
- `--sync-bn`: ใช้ synchronized Batch Normalization สำหรับการฝึกบน GPU หลายตัว (ถ้ามี)

- การปรับพารามิเตอร์:

- ในขั้นตอนแรกจะเริ่มฝึกด้วย YOLOv5s (Small)
- เนื่องจาก PC ของเรามีประสิทธิภาพสูง (สามารถรันได้ 200+ FPS) จึงมีความเป็นไปได้ที่จะลองปรับเป็น YOLOv5m (Medium) ในอนาคตเพื่อเพิ่มความแม่นยำ

### 3.3.2.2 การฝึก YOLOv5s\_v2 / m / l / x บน Dataset v2 (Training V2)

หลังปรับ Dataset (หัวข้อ 3.2.4) ให้ใหญ่และหลากหลายขึ้น ผู้วิจัยใช้ทรัพยากร V100 2 ตัว เทรนโมเดล 4 ขนาด

Model	Device (GPU)	Batch	Epochs (early-stop)	พารามิเตอร์	Command key-points
<b>yolov5s_v2</b>	--device 1 (V100-#2)	32	150 → หยุดที่ <b>109</b>	7.0 M params, 15.8 GFLOPs	train from scratch (เนื่องจาก weight ที่จะใช้เทรนดีใน window แต่ต้องย้ายมาใช้ใน Linux ทำให้เกิดความขัดแย้งกัน)
<b>yolov5m</b>	--device 0 (V100-#1)	24	200 → หยุดที่ <b>112</b>	20.9 M params, 47.9 GFLOPs	
<b>yolov5l</b>	--device 0	16	200 → หยุดที่ <b>58</b>	46.1 M params, 107 GFLOPs	
<b>yolov5x</b>	--device 1	12	200 → หยุดที่ <b>59</b>	87 M params, 218 GFLOPs	

Option ร่วมที่เปิดทุกโมเดล

--optimizer Adam # learning-rate ยึดหยุ่น

--sync-bn # batch norm ข้าม GPU

--rect # rectangular training ลด VRAM + เพิ่ม IoU

--patience 50 # early stop ถ้า val-mAP ไม่ดีขึ้น

เหตุผลเพิ่ม  $L$  และ  $X$  – เมื่อมี VRAM/เวลาเหลือจึงขยายการทดลอง เพื่อดูว่าความแม่นยำจะคุ้มค่า latency หรือไม่ในงาน real-time

### 3.3.3 [การประเมินผล โมเดล (Evaluation Metrics)]


#### 3.3.3.1 [ผลลัพธ์ YOLOv5s รุ่นแรก (Training V1)]

- ตัวชี้วัดหลักที่ใช้วัดผล:

- **Precision:** วัดความแม่นยำของการตรวจจับ (จำนวนการตรวจจับที่ถูกต้องต่อการตรวจจับทั้งหมด)
- **Recall:** วัดความสามารถในการจับเป้าหมายที่มีอยู่จริง
- **mAP (mean Average Precision):** ประเมินความแม่นยำเฉลี่ยของโมเดลในการตรวจจับวัตถุ
- **Confidence Score:** คะแนนความมั่นใจของโมเดลในการตรวจจับแต่ละกล่อง

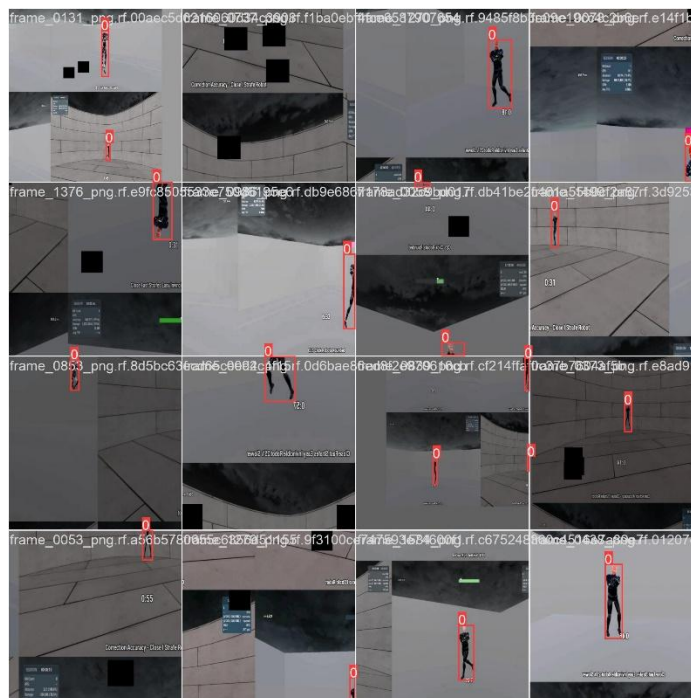
- **ผลการประเมินจาก YOLOv5s (Final Evaluation Results):**

จากผลลัพธ์การประเมินขั้นสุดท้ายของ YOLOv5s บนชุดทดสอบ (Validation Set) ได้ค่าชี้วัดดังนี้:

YOLOv5s summary: 157 layers, 7012822 parameters, 0 gradients, 15.8 GFLOPs							
Class	Images	Instances	P	R	mAP50	mAP50-95:	100%  
all	1042	2640	0.932	0.804	0.892	0.504	

- **Precision:** 0.932
- **Recall:** 0.804
- **mAP (mean Average Precision):** 0.892
- **Confidence Score:** 0.504
- ซึ่งค่าดังกล่าวแสดงให้เห็นว่าโมเดลมีความสามารถในการตรวจจับเป้าหมายในระดับที่น่าพอใจ โดยมีความแม่นยำในการจับวัตถุสูง และสามารถจับได้ในหลายสถานการณ์ที่หลากหลาย โดยเฉพาะ **mAP@0.5** ที่เกิน **0.85** ถือว่ามีความแม่นยำที่เหมาะสมสำหรับงานเบื้องต้น
  - อย่างไรก็ตาม ค่า **mAP@0.5:0.95** ยังมีพื้นที่ให้ปรับปรุง ซึ่งเป็นสิ่งที่คาดการณ์ไว้แล้วจากลักษณะของ Dataset ที่มีการ Label ผิดพลาดบางส่วน และการ Augment ที่อาจยังไม่ครอบคลุมความหลากหลายของ Scenarios ทั้งหมด

- **ผลลัพธ์ที่สังเกตได้:**



- จากการฝึกเบื้องต้น พบว่าโมเดลสามารถตรวจจับเป้าหมายในบางสถานการณ์ได้อย่างแม่นยำและไม่เกิดการตรวจจับผิด (False Positives)
- ในบางฉากที่มีความซับซ้อนหรือมีความเคลื่อนไหวรวดเร็ว โมเดลอาจพลาดเป้าหมาย (Missed Detections)
- ผลลัพธ์ที่แสดงในภาพตัวอย่าง (เช่น train result.png, train\_batch2.jpg) แสดงให้เห็นถึงการทำงานที่ดีในบางสถานการณ์ แต่ยังมีพื้นที่ให้ปรับปรุงความแม่นยำและความเสถียรในสถานการณ์ที่ท้าทาย
- แผนการปรับปรุง:
  - ปรับปรุง Dataset โดยตรวจสอบและเพิ่มจำนวนภาพในสถานการณ์ที่โมเดลพลาดเป้าหมาย
  - Fine-tune พารามิเตอร์การฝึกและพิจารณาการเปลี่ยนแปลงจาก YOLOv5s เป็น YOLOv5m เมื่อการผสมระบบฮาร์ดแวร์เสร็จสมบูรณ์

### 3.3.3.2 [ผลลัพธ์ Training V2 (Dataset v2)]

Metric	s_v2	m	l	x
Precision	0.885	0.843	0.821	0.844
Recall	0.652	0.602	0.681	0.681
mAP@0.5	0.789	0.747	0.761	0.770
mAP@0.5:0.95	0.381	0.313	0.256	0.290



Metric	s_v2	m	l	x
Inference (ms) @640	2.0	4.7	8.2	14.6

จากผลการฝึกและการประเมินโมเดล YOLOv5 ทั้ง 4 ขนาด (s\_v2, m, l, x) บน Dataset v2 สามารถสรุปข้อสังเกตได้ดังนี้:

- **YOLOv5s\_v2**

แสดงประสิทธิภาพโดยรวมดีที่สุดเมื่อพิจารณาทั้งด้าน **ความแม่นยำ (Accuracy)** และ **ความเร็วในการประมวลผล (Latency)**

- mAP@0.5: **0.789**
- Latency: **2.0 ms/frame**

เหมาะอย่างยิ่งสำหรับระบบที่ต้องการทำงานแบบเรียลไทม์ (real-time)

- **YOLOv5m**

แม้มีจำนวนพารามิเตอร์มากกว่า แต่พบว่า **Recall ลดลง** และมีแนวโน้มที่จะเกิด **Overfitting**

- mAP@0.5: **0.747**
- Recall: **0.602**
- Latency: **4.7 ms/frame**

- **YOLOv5l**

มี **Recall สูงสุด** ในบรรดาทั้งหมด

- Recall: **0.681**
- mAP@0.5: **0.761**
- Latency: **8.2 ms/frame**

เริ่มมีผลต่อความลื่นไหลของเกมในระบบที่มีข้อจำกัดด้านประสิทธิภาพ

- **YOLOv5x**

ให้ค่าความแม่นยำสูงใกล้เคียง s\_v2 แต่มี latency สูงสุด

- mAP@0.5: **0.770**
- Latency: **14.6 ms/frame**

ซึ่งอาจทำให้ระบบมี FPS ต่ำกว่าที่สามารถใช้งานได้จริงในเกมแนว FPS

### 3.3.4 ปัญหา & แนวทางแก้ไข

สำหรับการฝึกโมเดลรอบใหม่ (Training V2) ผู้วิจัยพบปัญหาเพิ่มเติมที่ไม่เคยเกิดขึ้นในรอบแรก โดยได้ดำเนินการวิเคราะห์และวางแนวทางในการแก้ไขไว้ดังนี้:

#### 3.3.4.1 ปัญหาในการใช้งาน Weight ระหว่าง Windows และ Linux

ปัญหา:

เดิมผู้วิจัยมีแผนที่จะนำ weight จาก Windows ที่ฝึกไว้ก่อนหน้านี้มาใช้ต่อบนระบบ Linux ที่ใช้เซิร์ฟเวอร์คอนเซ (Tesla V100 x2) แต่ขณะนั้นมีการฝึกโมเดลอีกตัวหนึ่งอยู่พร้อมกันบนอีก GPU และการจะฝึก “ต่อ” (continue

training) จำเป็นต้องแก้ไขไฟล์ train.py ซึ่งมีความเสี่ยงสูงที่จะกระทบกับกระบวนการฝึกของอีกโมเดลที่กำลังทำงานอยู่

ด้วยข้อจำกัดดังกล่าว ผู้วิจัยจึงตัดสินใจเริ่มฝึก YOLOv5s\_v2 ใหม่ทั้งหมด โดยใช้ pretrained weights พื้นฐาน (yolov5s.pt) แทนที่จะโหลด weights ที่เทรนไว้ก่อนหน้านี้

หลังจากนั้น เมื่อจะนำ weight ที่ฝึกเสร็จแล้วบนระบบ Linux มาใช้งานจริงบนเครื่อง PC (Windows) พบปัญหาเพิ่มเติม คือ torch.load() ไม่สามารถโหลด path ที่เซฟไว้ในรูปแบบ PosixPath ได้ในระบบ Windows ส่งผลให้เกิดข้อผิดพลาด KeyError: model.ema หรือ AttributeError: PosixPath object has no attribute 'read'

**แนวทางแก้ไข:**

เพื่อให้สามารถใช้งาน weight ได้บน Windows จำเป็นต้องเพิ่มคำสั่ง patch ที่หัวไฟล์ Python ก่อนเรียก torch.load() ดังนี้:

CopyEdit

```
import pathlib
```

```
pathlib.PosixPath = pathlib.WindowsPath
```

คำสั่งนี้จะช่วยให้ระบบ Windows สามารถแปลงและอ่าน path ที่บันทึกมาจากระบบ Linux ได้อย่างถูกต้อง

### 3.3.4.2 ปัญหา VRAM Overflow บน RTX 3060

**ปัญหา:**

ในการฝึกโมเดลขนาดกลาง (YOLOv5m) โดยใช้ Dataset v2 ที่มีจำนวนภาพเพิ่มขึ้น พบว่าเมื่อใช้ batch size มากกว่า 12 จะเกิดปัญหา CUDA out-of-memory (OOM) บนเครื่องที่ใช้ GPU RTX 3060

**แนวทางแก้ไข:**

ผู้วิจัยจึงตัดสินใจย้ายการฝึกทั้งหมดไปยังเซิร์ฟเวอร์คณะฯ ที่มี GPU Tesla V100 จำนวน 2 ตัว ซึ่งสามารถรองรับการฝึกโมเดลขนาดใหญ่ได้ดี และยังสามารถเปิด option --rect เพื่อช่วยลด memory footprint ในการจัด batch ขณะฝึกได้อีกด้วย

## 3.4[การออกแบบและพัฒนาฮาร์ดแวร์ (Hardware Design and Development)]

[เนื้อหา]

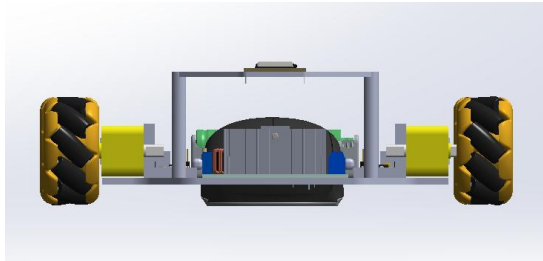
### 3.4.1 [รายละเอียดการออกแบบ CAD]

- เครื่องมือ:

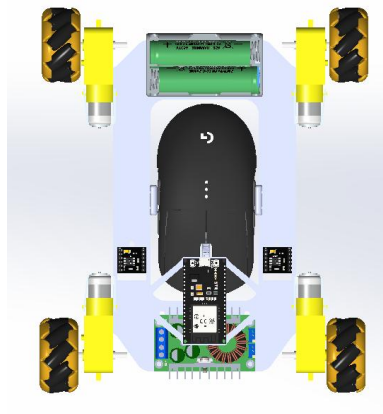
- ใช้โปรแกรม **SolidWorks** ในการออกแบบโครงสร้างหุ่นยนต์

- มุมมองการออกแบบ:

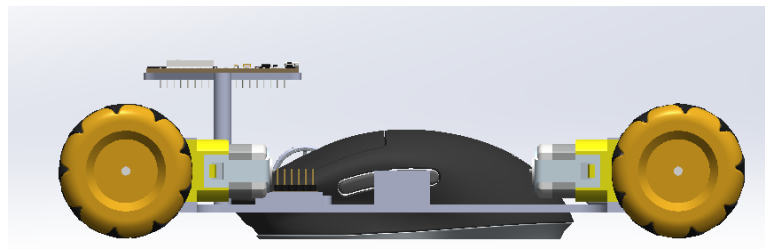
- มีการสร้างภาพจากมุมมองต่าง ๆ
- **Front View**



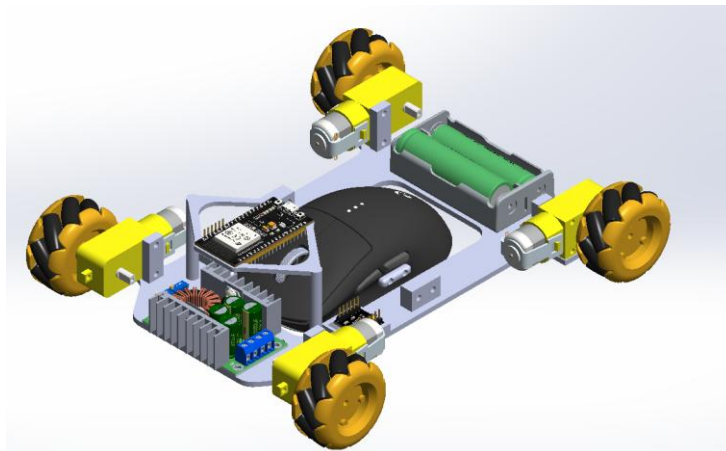
- **Top View**



- **Side View**



- **Isometric View**



- รายละเอียด:

- การออกแบบ CAD แสดงตำแหน่งของอุปกรณ์หลัก เช่น มอเตอร์, ล้อ Mecanum, ESP32, แบตเตอรี่, และส่วนประกอบ 3D Printed
- โดยมีการวางแผนระบบสองชั้น (First floor สำหรับการประกอบฮาร์ดแวร์หลัก และ Second floor สำหรับการติดตั้ง ESP32 เพื่อการจัดการสายไฟที่ดีขึ้น)

### 3.4.2 [รายการอุปกรณ์ที่ใช้ (Bill of Materials - BOM)]

รายการอุปกรณ์หลักที่ใช้ในโครงงานนี้มีดังนี้:

- แบตเตอรี่และอุปกรณ์จัดเก็บพลังงาน:
  - **1 × 2 Battery 18650 Rail:** ที่ใช้สำหรับติดตั้งแบตเตอรี่ 18650
  - **2 × 18650 Batteries:** แบตเตอรี่ลิเทียมสำหรับจ่ายพลังงานให้กับระบบ
- ล้อและการเคลื่อนที่:
  - **4 × Mecanum Wheels (48mm):** แบ่งเป็น 2 ล้อซ้ายและ 2 ล้อขวา สำหรับให้ความสามารถในการเคลื่อนที่แบบ omnidirectional
- อุปกรณ์เสริม:
  - **Logitech G Pro X Superlight Mouse:** ใช้เป็นตัวทดสอบการควบคุมการเล็ง (แม้จะไม่แน่ใจว่าต้องนับใน BOM หรือไม่ แต่รวมไว้ในรายละเอียดของระบบ)
- ไมโครคอนโทรลเลอร์และการสื่อสาร:
  - **ESP32 38-pin:** สำหรับการประมวลผลและสื่อสารข้อมูล (ทั้งผ่าน WiFi และ Bluetooth)
  - **Top ESP32 Mount (3D Printed):** ชุดที่ติดตั้งบนชั้นบนของระบบ เพื่อจัดการสายไฟและการเชื่อมต่อที่สะดวก
- การควบคุมแรงดันไฟฟ้า:
  - **XL4016E1 Step Down Converter:** ใช้ปรับแรงดันจากแบตเตอรี่ให้เหมาะสมกับระบบ
- มอเตอร์และไดรเวอร์:
  - **4 × Yellow TT Motors (Gear Ratio 1:48):** มอเตอร์ที่เลือกใช้เพื่อให้ได้ความเร็วในการเคลื่อนที่ที่สูงขึ้น (แม้แรงบิดจะต่ำกว่า แต่เพียงพอสำหรับระบบเมาส์ที่เบา)
  - **2 × DRV8833 Motor Drivers:** ใช้ควบคุมมอเตอร์ 4 ตัว (แต่ละไดรเวอร์ควบคุมมอเตอร์ 2 ตัว) เนื่องจากมีขนาดเล็ก น้ำหนักเบา และเหมาะสำหรับการใช้งานในระบบนี้
- ส่วนประกอบ 3D Printed:
  - **1 Base:** โครงสร้างหลักที่ถูกพิมพ์ 3D เพื่อรองรับการติดตั้งอุปกรณ์ทั้งหมด
  - **2 × Mouse Lock (3D Printed):** ชิ้นส่วนที่ช่วยยึดตำแหน่งเมาส์ให้คงที่ในขณะที่ใช้งาน

### 3.4.3 [กระบวนการประกอบชิ้นส่วนจริงตามแบบ CAD]

ในการประกอบชิ้นส่วนจริงจากแบบ CAD ผู้วิจัยได้ดำเนินการตามขั้นตอนที่ออกแบบไว้ แต่พบปัญหาบางประการที่สำคัญในการประกอบครั้งแรก ดังนี้:

#### 1. ปัญหาด้านการพิมพ์ 3D (วัสดุ PETG)

เนื่องจากเป็นครั้งแรกที่เครื่องพิมพ์ 3D ของผู้วิจัยใช้งานวัสดุ PETG ทำให้การตั้งค่าเครื่องพิมพ์ยังไม่เหมาะสม และเซ็นเซอร์วัดระยะ (Z-sensor) ของเครื่องพิมพ์มีปัญหา ส่งผลให้ชิ้นงานที่พิมพ์ออกมามีความแข็งแรงต่ำ มีลักษณะเปราะ หรืออ่อนตัวมากจนคล้ายสปริง ชิ้นงานบางชิ้นเกิดการบิดโค้ง ไม่ตรงตามที่ออกแบบไว้ ส่งผลให้เมื่อประกอบแล้วตัวหุ่นยนต์มีลักษณะโยกเยก ไม่มั่นคง

- ภาพแสดงชิ้นส่วนที่มีปัญหาการพิมพ์ (ครั้งแรก)  
[แทรกรูปภาพที่นี่]

---

#### 2. การปรับปรุงและการพิมพ์ชิ้นส่วนใหม่ (ครั้งที่สอง)

หลังจากที่พบปัญหาในการประกอบครั้งแรกแล้ว ผู้วิจัยได้ทำการปรับปรุงแบบ CAD และตั้งค่าเครื่องพิมพ์ 3D ใหม่ เพื่อแก้ไขปัญหาดังกล่าว ที่พบ ได้แก่:

- **ปรับปรุงขนาดรูยึดมอเตอร์:**  
เนื่องจากครั้งแรกยังไม่มั่นใจในขนาดของมอเตอร์ที่แน่นอน จึงออกแบบให้มีรูหลายจุดสำหรับการยึด หลังจากได้รับมอเตอร์จริง จึงปรับ CAD ใหม่ให้เหลือรูเฉพาะตำแหน่งที่จำเป็นเท่านั้น  
รูป
- **การปรับขนาดรูสำหรับ XL4016:**  
ตัว XL4016 มีจุดที่บัดกรีที่ยื่นออกมา ส่งผลให้ติดตั้งลงบนฐานได้ไม่สนิท ผู้วิจัยจึงออกแบบฐานใหม่ โดยเพิ่มส่วนที่คล้ายกับบูท (Spacer) เข้ามาเพื่อรองรับจุดบัดกรี ทำให้ XL4016 ยึดแน่นและเสถียรมากขึ้น  
รูป

---

#### 3. การตั้งค่าแรงดันจาก XL4016:

ผู้วิจัยตั้งค่าแรงดันไฟฟ้าเอาต์พุตของ XL4016 ให้คงที่ที่ **7.4 V** โดยเลือกแรงดันนี้เนื่องจากเป็นแรงดันที่เหมาะสมกับมอเตอร์ Yellow TT Motor และเป็นค่าแรงดันใกล้เคียงกับแรงดันแบตเตอรี่ลิเธียม 18650 สองก้อนอนุกรม เพื่อให้การจ่ายไฟมีความเสถียรและไม่ผันผวนมากเกินไป

---

#### 4. การเชื่อมต่อสายไฟและการควบคุมมอเตอร์:

ESP32 38-pin ที่ใช้ในการควบคุมระบบหุ่นยนต์ ถูกเชื่อมต่อกับ Motor Driver (DRV8833) จำนวน 2 ตัว โดย

แต่ละตัวจะควบคุมมอเตอร์จำนวน 2 ตัว (รวมทั้งหมด 4 มอเตอร์) การกำหนดพอร์ตที่เชื่อมต่อ ESP32 กับ DRV8833 มีดังนี้:

#### ESP32 Pin DRV8833 Driver Motor

GPIO 25, 26 Driver ตัวที่ 1 มอเตอร์ล้อหน้าซ้าย

GPIO 27, 14 Driver ตัวที่ 1 มอเตอร์ล้อหน้าขวา

GPIO 12, 13 Driver ตัวที่ 2 มอเตอร์ล้อหลังซ้าย

GPIO 32, 33 Driver ตัวที่ 2 มอเตอร์ล้อหลังขวา

หมายเหตุ: การเลือก GPIO ดังกล่าวเป็นการเลือกจากพอร์ตที่สะดวกต่อการเดินสายและเหมาะสมสำหรับ PWM ที่ใช้ควบคุมมอเตอร์ผ่าน DRV8833

---

#### 5. ภาพแสดงการประกอบเสร็จสมบูรณ์:

หลังจากปรับปรุงและประกอบใหม่เรียบร้อยแล้ว ระบบหุ่นยนต์มีลักษณะและการจัดวางอุปกรณ์ที่เหมาะสมและมีความแข็งแรงเพียงพอสำหรับการทดสอบการทำงานต่อไป

[ แทรกรูปภาพประกอบเสร็จสมบูรณ์ที่นี่ ]

### 3.5[การผสานระบบ (System Integration)]

#### 3.5.1 [กระบวนการประกอบชิ้นส่วนจริงตามแบบ CAD]

ผู้วิจัยได้พัฒนาโปรแกรมเบื้องต้นด้วยภาษา Python เพื่อทำการจับภาพหน้าจอจากโปรแกรม Kovaak และส่งภาพไปยังโมเดล YOLOv5 ที่ฝึกไว้สำหรับการตรวจจับตำแหน่งเป้าหมาย โดยใช้ไลบรารี เช่น `mss`, `cv2`, และ `torch` สำหรับการทดสอบเบื้องต้นโมเดลสามารถทำงานได้อย่างถูกต้องและไม่ตรวจจับเป้าหมายผิดพลาด

อย่างไรก็ตาม โค้ดดังกล่าวยังเป็นเพียงการทดสอบแยกเฉพาะส่วน Vision Module และยังไม่ได้เชื่อมต่อเข้ากับระบบ ESP32 หรือการควบคุมหุ่นยนต์จริง

#### 3.5.2 [กระบวนการประกอบชิ้นส่วนจริงตามแบบ CAD]

\*อยู่ระหว่างการพัฒนาและจะดำเนินการในช่วง Phase ถัดไป หลังจากการประกอบชิ้นส่วนฮาร์ดแวร์เสร็จสมบูรณ์\*

### 3.5.3 [กระบวนการประกอบชิ้นส่วนจริงตามแบบ CAD]

\*อยู่ระหว่างการพัฒนาและจะดำเนินการในช่วง Phase ถัดไป หลังจากการประกอบชิ้นส่วนฮาร์ดแวร์เสร็จสมบูรณ์\*

## 3.6[การผสานระบบ (System Integration)]

เนื่องจากระบบยังอยู่ในขั้นตอนการประกอบและพัฒนาโมดูลต่าง ๆ แยกส่วน ขั้นตอนการทดสอบระบบรวมจะเริ่มดำเนินการในช่วงถัดไป โดยแผนการทดสอบประกอบด้วย:

### 3.6.1 การทดสอบเบื้องต้น

- ทดสอบการหมุนของมอเตอร์แต่ละตัวจากการสั่งงานผ่าน ESP32
- ทดสอบระบบแปลงแรงดันจาก XL4016E1
- ทดสอบความแม่นยำในการตรวจจับของโมเดล YOLOv5 ด้วยภาพจากโปรแกรม Kovaak

### 3.6.2 การทดสอบระบบรวม

- เมื่อระบบหุ่นยนต์สามารถรับข้อมูลจากโมเดลและเคลื่อนที่ได้จริง จะทำการทดสอบโดยให้ระบบเล็งและยิงในโปรแกรม Kovaak

### 3.6.3 การเก็บข้อมูลและวิเคราะห์ผล

- คะแนนรวมจาก Kovaak
- Targets Hit / Targets Missed
- Accuracy และ Percent Hit

## บทที่ 4 การทดลองและผลการทดลอง/วิจัย

ในการทดลองครั้งนี้ ผู้วิจัยได้แบ่งการทดลองออกเป็น 3 ส่วนหลักเพื่อให้ง่ายต่อการวิเคราะห์และปรับปรุงระบบ ได้แก่

- ส่วนที่ 1: การทดสอบซอฟต์แวร์ (Software Testing)
- ส่วนที่ 2: การทดสอบฮาร์ดแวร์ (Hardware Testing)
- ส่วนที่ 3: การทดสอบระบบรวม (Integrated System Testing)

### 4.1[การทดสอบซอฟต์แวร์ (Software Testing)]

[ในขั้นตอนการทดสอบซอฟต์แวร์นี้จะแบ่งออกเป็น 2 หัวข้อย่อย คือ การทดสอบความสามารถในการตรวจจับวัตถุของโมเดล (Object Detection Performance) และการทดสอบอัตราเฟรมในการประมวลผลภาพของโมเดล (Frame Rate Testing)]

#### 4.1.1 [การทดสอบความสามารถในการตรวจจับวัตถุจากวิดีโอ (Object Detection Performance)]

ผู้วิจัยได้ทำการบันทึกวิดีโอจากโปรแกรม Kovaak ที่ความละเอียดระดับ Full HD (1920x1080) และอัตราเฟรม 12 fps โดยแต่ละ scenario จะมีความยาว 10 วินาที รวมทั้งหมด 5 scenarios ดังนี้

ชื่อ Scenario	รายละเอียด	จำนวนเฟรมทั้งหมด	ตรวจพบวัตถุ (เฟรม)	ไม่พบวัตถุ (เฟรม)
Close Fast Colosseum Robots No Shooting	Robot วิ่งรอบ ๆ ไม่มี Damage	120	120	0
[N] CLS Click Robots	Robot วิ่งรอบ ๆ มี Damage	119	119	0
RoboTS180	Robot หลายตัววิ่งรอบ ๆ มี Damage	119	322	3
Cata IC Fast Strafes Robot	Robot วิ่ง มี Damage, สภาพแวดล้อมต่างออกไป	119	108	11
Close Fast Strafes Invincible OW Robot	Robot วิ่งใกล้มาก ไม่มี Damage	120	120	0



ในการทดลองนี้ได้ใช้ Python (frame\_count.py) ในการแสดงแต่ละเฟรมเพื่อทำการนับจำนวนวัตถุด้วยตนเอง (Manual Counting) เพื่อให้มั่นใจว่าข้อมูลที่ได้ถูกต้องที่สุด โดยสคริปต์จะสรุปผลออกมาเป็นจำนวนเฟรมทั้งหมดที่ตรวจพบวัตถุและจำนวนเฟรมที่ไม่พบวัตถุอย่างชัดเจน

โค้ดที่ใช้ในการทดสอบ (frame\_count.py) เป็นดังนี้:

```
1 import cv2
2
3 def main():
4     # === EDIT THESE PATHS/PARAMETERS ===
5     video_path = "RoboTS180.mp4" # <-- Replace with your video file path
6     skip_frames = 1 # Change to N to sample every Nth frame (e.g., 2 to see every other frame)
7     # =====
8
9     cap = cv2.VideoCapture(video_path)
10    if not cap.isOpened():
11        print(f"Error: cannot open video {video_path}")
12        return
13
14    frame_idx = 0
15    counts = []
16
17    while True:
18        ret, frame = cap.read()
19        if not ret:
20            break
21        if frame_idx % skip_frames != 0:
22            frame_idx += 1
23            continue
24
25        cv2.imshow("Frame", frame)
26        cv2.waitKey(1) # Necessary to render the window
27
28        # Prompt user for manual count input
29        while True:
30            val = input(f"Frame {frame_idx}: Enter detected robot count (or 'q' to quit): ")
31            if val.lower() == 'q':
32                cap.release()
33                cv2.destroyAllWindows()
34                summary(counts)
35                return
36            try:
37                count = int(val)
38                break
39            except ValueError:
40                print("Invalid input. Please enter an integer.")
41
42        counts.append(count)
43        frame_idx += 1
44
45    cap.release()
46    cv2.destroyAllWindows()
47    summary(counts)
48
49
50 def summary(counts):
51     total_frames = len(counts)
52     total_detections = sum(counts)
53     false_negatives = sum(1 for c in counts if c == 0)
54
55     print("\n=== Summary ===")
56     print(f"Frames counted: {total_frames}")
57     print(f"Total detections: {total_detections}")
58     print(f"Average detections per frame: {total_detections/total_frames if total_frames else 0:.2f}")
59     print(f"False negatives (frames with zero detections): {false_negatives}")
60
61 if __name__ == "__main__":
62     main()
```

หลังจากทำการนับด้วยตนเอง (Manual Counting) เสร็จแล้ว ผู้วิจัยได้ทดสอบโมเดลทั้ง 5 ขนาด (small\_old, small\_new, medium, large, x) โดยใช้สคริปต์ Python ที่พัฒนาขึ้นเพื่อวิเคราะห์ความสามารถของแต่ละโมเดลในการตรวจจับเป้าหมายจากวิดีโอจริงของโปรแกรม Kovaak

โค้ดที่ใช้มีชื่อว่า `modeltest.py` ซึ่งทำหน้าที่โหลดวิดีโอจากแต่ละ Scenario และรันการตรวจจับแบบ `frame-by-frame` โดยวัดค่าดังต่อไปนี้:

- จำนวนเฟรมทั้งหมด
- จำนวนเฟรมที่ตรวจจับได้
- จำนวนเฟรมที่ไม่ตรวจจับ
- จำนวนวัตถุทั้งหมดที่ตรวจพบ (Total Detections)
- ค่าเฉลี่ย Confidence (Average Confidence Score)

```

import os
import pathlib
# ← Patch PosixPath → WindowsPath so torch.load can reconstruct paths on
Windows
pathlib.PosixPath = pathlib.WindowsPath

import yolov5
import cv2

# === EDIT THESE PATHS & PARAMETERS ===
VIDEO_DIR = r"D:\UNIVERSITY\YR3\FRA361_Open_Topic\Experience\modeltest"
VIDEO_FILES = [
    "[N] CLS Click Robots.mp4",
    "Cata IC Fast Strafes Robot.mp4",
    "Close Fast Colosseum Robots No Shooting.mp4",
    "Close Fast Strafes Invincible OW Robot.mp4",
    "Robot5180.mp4"
]

MODEL_CONFIGS = [
    {"name": "small_old",
     "weights": r"model\model1(y5s)\weights\best.pt",
     "imgsz": 640},
    {"name": "small_new",
     "weights": r"model\model2(y5s)\yolov5s_best.pt",
     "imgsz": 640},
    {"name": "medium",
     "weights": r"model\model3(y5m)\yolov5m_best.pt",
     "imgsz": 640},
    {"name": "large",
     "weights": r"model\model4(y5l)\yolov5l_best.pt",
     "imgsz": 640},
    {"name": "x",
     "weights": r"model\model5(y5x)\yolov5x_best.pt",
     "imgsz": 640},
]

CONF_THRESHOLD = 0.25 # minimum confidence
SKIP_FRAMES = 1 # process every Nth frame (1 = every frame)
# =====

def analyze_video(model, video_path, imgsz):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f" X could not open {os.path.basename(video_path)}")
        return None

    total_frames = detected_frames = no_detection_frames = total_detections
    = 0
    sum_confidences = 0.0
    frame_idx = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        if frame_idx % SKIP_FRAMES != 0:
            frame_idx += 1
            continue

        total_frames += 1
        results = model(frame, size=imgsz)
        preds = results.pred[0]

        nbox = preds.shape[0]
        if nbox > 0:
            detected_frames += 1
            total_detections += nbox
            sum_confidences += float(preds[:, 4].sum())
        else:
            no_detection_frames += 1

        frame_idx += 1

    cap.release()
    avg_conf = (sum_confidences / total_detections) if total_detections else
    0.0
    return {
        "total_frames": total_frames,
        "detected_frames": detected_frames,
        "no_detection_frames": no_detection_frames,
        "total_detections": total_detections,
        "avg_confidence": avg_conf,
    }

def main():
    for cfg in MODEL_CONFIGS:
        print(f"\n=== Model: {cfg['name']} ===")
        model = yolov5.load(cfg["weights"])
        model.conf = CONF_THRESHOLD

        for vid in VIDEO_FILES:
            path = os.path.join(VIDEO_DIR, vid)
            stats = analyze_video(model, path, cfg["imgsz"])
            if stats is None:
                continue

            print(f"Video: {vid}")
            print(f" • Total frames : {stats['total_frames']}")
            print(f" • Detected frames : {stats['detected_frames']}")
            print(f" • No-detect frames : {stats['no_detection_frames']}")
            print(f" • Total detections : {stats['total_detections']}")
            print(f" • Avg. confidence : {stats['avg_confidence']:.2f}")

if __name__ == "__main__":
    main()

```

โดยโค้ดจะใช้โมเดล YOLOv5 แต่ละขนาดที่ผ่านการฝึกเสร็จแล้ว และวิเคราะห์ภาพด้วยอัตรา SKIP\_FRAMES = 1 (วิเคราะห์ทุกเฟรม)

ผลลัพธ์การตรวจจับสามารถสรุปเป็นตารางดังนี้:

Scenario	small_old	small_new	medium	large	x
	Det/Total (rate%)	Det/Total (rate%)	Det/Total (rate%)	Det/Total (rate%)	Det/Total (rate%)
<b>[N] CLS</b>					
<b>Click</b>	87 / 119 (73.1%)	88 / 119 (74.0%)	111 / 119 (93.3%)	119 / 119 (100.0%)	119 / 119 (100.0%)
<b>Robots</b>					
<b>Cata IC</b>					
<b>Fast</b>	95 / 119 (79.8%)	32 / 119 (26.9%)	37 / 119 (31.1%)	69 / 119 (58.0%)	46 / 119 (38.7%)
<b>Strafes</b>					
<b>Robot</b>					
<b>Close Fast</b>					
<b>Colosseum</b>	72 / 120 (60.0%)	111 / 120 (92.5%)	118 / 120 (98.3%)	118 / 120 (98.3%)	119 / 120 (99.2%)
<b>Robots No</b>					
<b>Shooting</b>					
<b>Close Fast</b>					
<b>Strafes</b>	1 / 120 (0.8%)	118 / 120 (98.3%)	120 / 120 (100.0%)	120 / 120 (100.0%)	120 / 120 (100.0%)
<b>Invincible</b>					
<b>OW Robot</b>					
<b>RoboTS180</b>	101 / 119 (84.9%)	44 / 119 (37.0%)	23 / 119 (19.3%)	108 / 119 (90.8%)	45 / 119 (37.8%)

ค่าเฉลี่ย Confidence (ประมาณ):

- **small\_old:** 0.52
- **small\_new:** 0.43
- **medium:** 0.47
- **large:** 0.57
- **x:** 0.54

การเปรียบเทียบนี้ช่วยให้เข้าใจพฤติกรรมของโมเดลในสถานการณ์จริงได้ชัดเจนขึ้น ซึ่งการเลือกโมเดลที่เหมาะสมกับระบบต้องอิงจากทั้งความแม่นยำ และ latency ที่ยอมรับได้สำหรับ real-time application

## วิเคราะห์ผล

### 1. [N] CLS Click Robots

- ทุกโมเดล ทำได้ดี ( $\geq 73\% \rightarrow 100\%$ ) เพราะเป้าหมายเด่นชัด ไม่มีการ overlap หรือเคลื่อนไหวเร็วเกินไป

### 2. Cata IC Fast Strafes Robot

- small\_old (79.8%) ทำได้ดีเพราะ generalize จาก data หลาย environment
- small\_new (26.9%) & medium (31.1%) ต่ำ เพราะ dataset v2 มีเงื่อนไขที่แตกต่าง (damage, bg ต่างกัน) โมเดลเก่าที่ผ่าน augment ไม่ครอบคลุม
- large (58.0%) & x (38.7%) พอปรับขนาดช่วย detect บางกรณี แต่ยัง struggle ตอน overlap หรือความเร็วสูง

### 3. Close Fast Colosseum Robots No Shooting

- small\_old เพียง 60.0% เนื่องจากมุมกล้อง/background คล้ายกันจนโมเดลแรกสับสน
- โมเดลใหม่ (small\_new  $\rightarrow$  x) ปรับ augment และ data เพิ่ม ทำได้  $\geq 92.5\%$

### 4. Close Fast Strafes Invincible OW Robot

- small\_old แทบจับไม่ได้ (0.8%) เพราะ target โกลิ้งมากจนขอบ crop หลุด
- ทุกโมเดลใหม่ detect ได้ดี ( $\geq 98\%$ ) หลังจากเพิ่ม augment การโกลิ้งมุกกล้อง

### 5. RoboTS180

- small\_old (84.9%) ทำได้ดีในการ detect หลายตัว แต่บางเฟรม confidence สูงค้าง
- small\_new & medium & x ต่ำมาก (19–37%) เพราะ multiple robots overlap กัน โมเดลเสีย recall
- large (90.8%) ทำดีที่สุด เนื่องจาก capacity สูงพอแยก object ซ้อนกัน

## สรุปภาพรวม:

- โมเดลขนาดใหญ่ (large, x) ยังทำงานได้ดีที่สุดในกรณีที่มีจำนวน object มากหรือ overlap กัน (RoboTS180)
- small\_old & small\_new เหมาะกับ scenario เป้าหมายเดี่ยวหรือ BG เปลี่ยน แต่พลาดในกรณี extreme camera view
- medium ให้ผลลัพธ์ผสม ๆ ควรปรับ data augment เพิ่มเติมเพื่อรองรับ scenario พิเศษ

1. เนื้อหา

2. เนื้อหา

#### 4.1.2 [การทดสอบ Latency บน RTX 3060 (Inference Latency Testing)]

##### วัตถุประสงค์

วัดเวลาที่ใช้ในการประมวลผลแต่ละเฟรม (inference time) ของโมเดล YOLOv5 ขนาดต่างๆ บนเครื่อง ASUS TUF A15 + RTX 3060 Laptop (Max-Q) โดยใช้วิดีโอ 5 Scenario เดิม เพื่อประเมินว่าความหน่วง (latency) ต่อเฟรมอยู่ในเกณฑ์ที่รับได้หรือไม่

##### วิธีการทดลอง

1. ใช้สคริปต์ Python (modeltest\_latency.py) ทำงานดังนี้:
  - โหลดโมเดล YOLOv5 (small\_old, small\_new, medium, large, x) และตั้งค่า confidence threshold = 0.25
  - เปิดอ่านวิดีโอแต่ละ Scenario (12 fps,  $\approx 120$  เฟรม) ด้วย OpenCV
  - สำหรับแต่ละเฟรม:
    1. บันทึกเวลาเริ่มต้น ( $t_0$ ) ด้วย time.time()
    2. รัน inference model(frame)
    3. บันทึกเวลาสิ้นสุด ( $t_1$ )
    4. คำนวณ latency เฟรมปัจจุบัน =  $t_1 - t_0$
  - สรุปค่า latency เฟรมเฉลี่ย (mean), ค่าน้อยสุด (min) และค่ามากที่สุด (max) แต่ละ Scenario และแต่ละโมเดล

```

import os
import pathlib
import time

# Monkey-patch to avoid PosixPath unpickle errors on Windows
import pathlib as _pl
_pl.PosixPath = _pl.WindowsPath

import yolov5
import cv2

# === EDIT THESE PATHS & PARAMETERS ===
VIDEO_DIR = r"D:\UNIVERSITY\YR3\FRA361_Open_Topic\Experience\modeltest"
VIDEO_FILES = [
    "[N] CLS Click Robots.mp4",
    "Cata IC Fast Strafes Robot.mp4",
    "Close Fast Colosseum Robots No Shooting.mp4",
    "Close Fast Strafes Invincible OW Robot.mp4",
    "Robots180.mp4"
]

MODEL_CONFIGS = [
    {"name": "small_old", "weights": r"model\model1(y5s)\weights\best.pt",
     "imgsz": 640},
    {"name": "small_new", "weights": r"model\model2(y5s)\yolov5s_best.pt",
     "imgsz": 640},
    {"name": "medium", "weights": r"model\model3(y5m)\yolov5m_best.pt",
     "imgsz": 640},
    {"name": "large", "weights": r"model\model4(y5l)\yolov5l_best.pt",
     "imgsz": 640},
    {"name": "x", "weights": r"model\model5(y5x)\yolov5x_best.pt",
     "imgsz": 640},
]

CONF_THRESHOLD = 0.25 # minimum confidence
SKIP_FRAMES = 1 # process every Nth frame (1 = every frame)
# =====

def analyze_video(model, video_path, imgsz):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"X could not open {os.path.basename(video_path)}")
        return None

    latencies = []
    frame_idx = 0

    # we only need timing, no need to count detections here
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        if frame_idx % SKIP_FRAMES != 0:
            frame_idx += 1
            continue

        t0 = time.time()
        _ = model(frame, size=imgsz) # inference
        t1 = time.time()

        latencies.append((t1 - t0) * 1000.0) # ms
        frame_idx += 1

    cap.release()
    if not latencies:
        return None

    return {
        "frames": len(latencies),
        "mean_ms": sum(latencies) / len(latencies),
        "min_ms": min(latencies),
        "max_ms": max(latencies),
    }

def main():
    print(f"Running inference timing on {len(VIDEO_FILES)} videos x {len(MODEL_CONFIGS)} models...\n")
    for cfg in MODEL_CONFIGS:
        print(f"=== Model: {cfg['name']} ===")
        model = yolov5.load(cfg["weights"])
        model.conf = CONF_THRESHOLD

        for vid in VIDEO_FILES:
            path = os.path.join(VIDEO_DIR, vid)
            stats = analyze_video(model, path, cfg["imgsz"])
            if stats is None:
                continue

            print(f"[vid:40s] frames: {stats['frames']:3d} "
                  f"mean: {stats['mean_ms']:.1f} ms "
                  f"min: {stats['min_ms']:.1f} ms "
                  f"max: {stats['max_ms']:.1f} ms")

        print()

if __name__ == "__main__":
    main()

```

2. ทำซ้ำสำหรับโมเดลทั้ง 5 ขนาด บนสภาพแวดล้อมเดียวกัน (Conda + Python 3.8, PyTorch 2.1, CUDA 12.1)

#### ผลการทดลอง (ms/frame)

Model	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Mean	Min	Max
small_old	16.8	15.1	15.8	14.0	15.9	<b>15.5</b>	11.6	185.7 <sup>1</sup>
small_new	15.7	14.8	16.2	16.1	15.2	<b>15.6</b>	11.5	34.0
medium	19.7	17.7	19.5	19.4	18.0	<b>18.9</b>	14.0	90.5
large	23.8	22.1	22.3	23.5	23.1	<b>23.0</b>	17.5	92.5
x	32.2	29.9	30.9	31.3	29.5	<b>30.8</b>	26.0	109.5

<sup>1</sup> ค่า Max ของ small\_old เกิดจากเฟรมกระโดดผิดปกติ (spike) เมื่อ GPU มีงานเบื้องหลัง

#### วิเคราะห์ผลเบื้องต้น

- **small\_old vs. small\_new**
  - ค่าเฉลี่ย latency ใกล้เคียงกัน (~15.5 ms)
  - small\_new มี Max spike น้อยกว่า (34 ms vs. 185 ms) แสดงว่าการปรับ dataset ช่วยลด outlier ได้
- **medium**
  - latency ~18.9 ms/frame (~53 FPS) เหมาะกับเกมที่ต้องการ  $\geq 50$  FPS
  - Max spike เกิดขณะประมวลผลภาพซับซ้อน (damage+many objects)
- **large**
  - latency ~23.0 ms/frame (~43 FPS) ยังพอเล่นได้ แต่เริ่มเห็นผลกระทบต่อความลื่นไหล
- **x**
  - latency ~30.8 ms/frame (~32 FPS) ค่อนข้างต่ำกว่าเกณฑ์ FPS ที่ต้องการจริง ( $\geq 60$  FPS)

#### ข้อสังเกต:

- ทุกโมเดลมีค่า Min อยู่ที่ ~11–14 ms แสดงว่า GPU ทำ inference ได้เร็วเมื่อภาพไม่ซับซ้อน
- ค่า Max มักเกิดขณะเจอภาพที่มี object ซับซ้อนหรือ background ซับซ้อน (เช่น RoboTS180)
- small\_new เป็นตัวเลือกที่สมดุล: latency ต่ำสุด, outlier น้อย, และ accuracy ดีขึ้นชัดเจน

### 4.1.3 [การทดสอบผลกระทบต่ออัตราเฟรม-เรตของเกม (In-game FPS Impact Test)]

#### วัตถุประสงค์

ทดสอบว่าโมเดล YOLOv5 แต่ละขนาดใช้ทรัพยากร GPU มาก-น้อยเพียงใด เมื่อรันร่วมกับ Kovaak แบบ real-



time บนเครื่อง ASUS TUF A15 (RTX 3060 Laptop Max-Q, 6144 MiB) เป้าหมายคือดูว่า FPS ของเกม ลดลงเท่าใด และตรวจสอบว่าโมเดลใหญ่กว่าจะสร้าง “คอขวด” หรือไม่

วิธีการทดสอบคือการเปิดใช้งานโมเดลแต่ละตัวในขณะที่เล่น scenario ของ Kovaak และบันทึกอัตราเฟรมเฉลี่ย ระหว่างการใช้งานโมเดล

โมเดลที่ใช้ทดสอบ:

1. YOLOv5 Small (ตัวเก่า)
2. YOLOv5 Small (ตัวใหม่ที่ได้ตรวจและเพิ่มเติม Dataset)
3. YOLOv5 Medium
4. YOLOv5 Large
5. YOLOv5 X-Large

ขั้นตอนทดลอง

1. เปิด Kovaak ที่ความละเอียด 1920 × 1080 (V-sync off) แล้วบันทึกค่า FPS ผ่านตัววัดในเกม
2. สลับโหลดโมเดล 5 ขนาดด้วยสคริปต์ modeltest\_runwithgame.py (ยัด confidence 0.25, imgsz 640)
3. วัดค่า FPS เฉลี่ยของเกม 30 วินาที / Scenario (เล่น Scenario เดิมเพื่อความคงที่)
4. บันทึก GFLOPs ของโมเดลจากสรุปของ YOLOv5 (model.summary()) เพื่อใช้เทียบภาระคำนวณ

ตารางผลการทดสอบ (RTX 3060 Laptop)

Model	GFLOPs	Avg. FPS in game*	สังเกตการตรวจจับจาก Aimbot View**
YOLOv5s (old)	15.8	274	กระตุกและตรวจไม่ค่อยเจอ
YOLOv5s (v2)	15.8	267	ตรวจเจอดีกว่าตัวเก่า แต่ก็ยังไม่ค่อยเจอเมื่อเร็วมากๆ
YOLOv5m	48.2	243	นิ่งกว่าตัวก่อนหน้าเยอะมาก
YOLOv5l	108.2	252	แทบจะไม่พลาดในการตรวจสอบเลย
YOLOv5x	204.6	247	ใกล้เคียงแทบไม่เห็นความต่างจากตัวก่อนหน้า

\* ค่า FPS เฉลี่ยเมื่อรันพร้อม YOLOv5 โดยไม่เปิดโมเดล เกมสามารถทำได้ประมาณ 270–290 FPS

\*\* Aimbot View คือหน้าต่างที่แสดงผลจากการตรวจจับของ YOLO บนภาพเกมเนื้อหา

### วิเคราะห์ผลเบื้องต้น

- แม้ว่า GFLOPs ของ YOLOv5x จะสูงถึง 204.6 แต่ค่า FPS ของเกมลดลงไม่มาก (~8–10%) สะท้อนว่า GPU ยังมีพลังงานเหลือสำหรับ render เกม
- ส่วนที่เห็นผลกระทบชัดเจนที่สุดคือ Aimbot View ซึ่งมีอาการ "กระตุก" เพิ่มขึ้นตามขนาดของโมเดล อันเนื่องจากต้อง copy frame และ render overlay เพิ่มเติมจาก YOLO
- YOLOv5l ให้ผลลัพธ์ที่สมดุลระหว่าง FPS และ Accuracy โดยตรวจจับได้แม่นยำมากแต่ยังคง FPS ใกล้เคียง YOLOv5s\_v2
- YOLOv5m แม้ FPS ต่ำกว่าเล็กน้อย แต่ภาพใน Aimbot View หนึ่งกว่า YOLOv5s\_v2 อย่างเห็นได้ชัด
- YOLOv5s\_v2 ยังคงเป็นโมเดลที่ประหยัดทรัพยากรที่สุด และสามารถตรวจจับได้ดีใน Scenario ทั่วไป แต่จะเริ่มมีปัญหาเมื่อมีการเคลื่อนไหวเร็วมากหรือมีเป้าหมายซ้อนกัน

## 4.2[หัวข้อ]

[เนื้อหา]

## 4.3[หัวข้อ]

[เนื้อหา]

## บทที่ 5 บทสรุป

[เนื้อหา]

### 5.1[หัวข้อ]

[เนื้อหา]

#### 5.1.1 [หัวข้อย่อย]

1. เนื้อหา
2. เนื้อหา

### 5.2[หัวข้อ]

[เนื้อหา]

เอกสารอ้างอิง