



โครงการ

การประมาณอัตราการเต้นหัวใจจากวิดีโอใบหน้าแบบเวลาจริง
Real-time Estimated Heart Rate From Facial Video

โดย

ด.ช.อศิระ ชาคาชิตะ วรเศวต

เสนอ

อ.ดร.โพธิวัฒน์ งามขจรวิวัฒน์

ดร.อภิชาติ อินทรพานิชย์

โครงการนี้เป็นส่วนหนึ่งของการเข้าร่วม

โครงการพัฒนาอัจฉริยภาพทางวิทยาศาสตร์และเทคโนโลยี

สำหรับเด็กและเยาวชน รุ่นที่ 25 ประจำปี 2565

และได้รับทุนทำโครงการวิทยาศาสตร์ จาก มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าธนบุรี

หัวข้อโครงการ	Real-time Estimated Heart Rate From Facial Video
ผู้เขียน	ด.ช.อศิระ ซาคาซิตะ วรเศวต
อาจารย์ที่ปรึกษา	อ.ดร.โพธิวัฒน์ งามขจรวิวัฒน์ ดร.อภิชาติ อินทรพานิชย์
โครงการ	พัฒนาอัจฉริยภาพทางวิทยาศาสตร์และเทคโนโลยีสำหรับเด็กและเยาวชน
ปีการศึกษา	2565

บทคัดย่อ

เนื่องจากผู้เขียนเห็นงานวิจัยที่สามารถประมาณอัตราการเต้นหัวใจจากวิดีโอใบหน้าได้โดยงานวิจัยที่เห็นส่วนใหญ่จะเป็นการนำวิดีโอจากฐานข้อมูลมาใช้ในการแสดงความถูกต้องของค่าที่ประมาณได้ แต่ผู้เขียนยังไม่เห็นการทดสอบแบบเวลาจริงจึงอยากทำการลองทดลองดูว่าการวัดอัตราการเต้นของหัวใจโดยไม่มีการสัมผัสแบบเวลาจริงเป็นไปได้หรือไม่ ผู้เขียนได้นำวิธีการที่นำเสนอในบทความวิจัยและมีคำสั่งที่เขียนด้วยภาษา Python ไว้แล้วมาประยุกต์ใช้กับคอมพิวเตอร์ และ Raspberry Pi เพื่อศึกษาความสามารถในการคำนวณค่าประมาณของอัตราการเต้นหัวใจแบบเวลาจริง เนื่องจากการประมวลผลวิดีโอในแต่ละเฟรม (frame) ด้วยคอมพิวเตอร์ หรือ Raspberry Pi ใช้เวลามาก ซึ่งจากการทดลองจะมีค่าประมาณ 10 FPS และ 5 FPS ตามลำดับซึ่งช้ากว่าความเร็วในการถ่ายภาพของกล้อง (30FPS) มากทำให้การคำนวณค่าประมาณของอัตราการเต้นหัวใจแบบเวลาจริงมีความผิดพลาดมากกว่าที่งานวิจัยได้นำเสนอไว้ และการประมวลผลแบบเวลาจริงจะต้องพิจารณาถึงเวลาที่แต่ละเฟรมถูกเก็บมาอาจไม่เท่ากัน จะต้องมีการปรับปรุงข้อมูลเพื่อให้ช่วงเก็บเวลาเก็บข้อมูลเท่าๆกัน โครงการที่พัฒนาขึ้นสามารถใช้งานบนคอมพิวเตอร์ได้จริงโดยยังมีการคลาดเคลื่อนอยู่ แต่ไม่สามารถทำได้ใน Raspberry Pi

สารบัญ

บทที่

1. บทนำ	1
1.1 ที่มาและความสำคัญ	1
1.2 วัตถุประสงค์	1
1.3 ขอบเขตของการดำเนินงาน	1
1.4 ประโยชน์ที่คาดว่าจะได้รับ	1
2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง	2
2.1 Photoplethysmogram (PPG)	2
2.2 Remote Photoplethysmography (rPPG)	2
2.3 Plane-Orthogonal-to-Skin (POS)	3
2.4 การแปลงฟูเรียร์ (Fourier Transform)	3
2.5 การกรองความถี่ในช่วงที่สนใจ (Bandpass Filter)	4
2.6 อัตราการเต้นของหัวใจ	4
2.7 การติดตามบริเวณของใบหน้า	4
3. การออกแบบโปรแกรม	6
3.1 ภาพรวมของโปรแกรม	6
3.2 ส่วนถ่ายและเก็บภาพ	6
3.3 ส่วนเก็บข้อมูล	7
3.4 ส่วนประมวลผลภาพ	9
3.5 การนำเข้าคำสั่งที่ต้องใช้	11
3.6 ส่วนเก็บภาพ	11
3.7 ส่วนเก็บข้อมูล	12
3.8 ส่วนประมวลผลภาพ	14

4. การทดลองและผลการทดลอง	17
4.1 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 25 เซนติเมตร	17
4.2 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 50 เซนติเมตร	17
4.3 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 75 เซนติเมตร	18
4.4 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 100 เซนติเมตร	18
4.5 ทดลองวัดอัตราการเต้นของหัวใจโดยใช้ภาพถ่าย	18
5. สรุปโครงงานและข้อเสนอแนะ	19
5.1 สรุปผลโครงงาน	19
5.2 ข้อเสนอแนะ	19
6. บรรณานุกรม	20

1 : บทนำ

1.1 ที่มาและความสำคัญ

อัตราการเต้นของหัวใจมีประโยชน์และการใช้งานมากมาย เช่นช่วยให้บุคลากรทางการแพทย์สามารถวินิจฉัยโรคหรือความผิดปกติทางร่างกายได้ และช่วยให้สามารถตรวจจับการโกหกในเวลาสอบสวนคดีได้ เป็นต้น แต่ปัจจุบันในการวัดอัตราการเต้นของหัวใจต้องมีการสัมผัสกับเซ็นเซอร์ซึ่งทำให้ขยับแขนได้ยาก อึดอัด และถ้าทำความสะอาดไม่ดีอาจทำให้เกิดโรคติดต่อได้ ผมจึงคิดที่จะทำโครงการนี้เพื่อที่จะทำให้สามารถวัดอัตราการเต้นของหัวใจได้สะดวกขึ้นและไม่เสี่ยงต่อการติดโรค

1.2 วัตถุประสงค์

เพื่อให้สามารถวัดอัตราการเต้นของหัวใจได้โดยไม่ต้องมีการสัมผัสแบบเวลาจริง

1.3 ขอบเขตของการดำเนินงาน

โครงการนี้มีจุดมุ่งหมายเพื่อสร้างโปรแกรมที่สามารถวัดอัตราการเต้นของหัวใจได้โดยไม่ต้องมีการสัมผัสกับอุปกรณ์ใดๆ

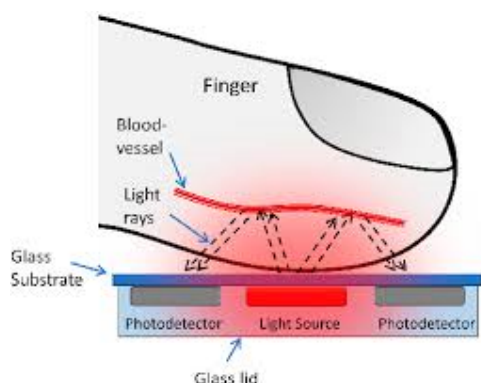
1.4 ประโยชน์ที่คาดว่าจะได้รับ

สามารถวัดอัตราการเต้นของหัวใจได้โดยไม่ต้องอึดอัดและไม่เสี่ยงที่จะติดโรคติดต่อ

2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 Photoplethysmogram (PPG)

หลักการของ PPG คืออุปกรณ์จะปล่อยแสงไปยังผิวหนังสะท้อนเส้นเลือดไปยังเซ็นเซอร์เพื่อตรวจจับปริมาณแสงที่สะท้อนเส้นเลือด ปริมาณของแสงเปลี่ยนแปลงไปตามปริมาตรของเลือด ซึ่งเกิดจากการขยายและการหดตัวของเส้นเลือดฝอย จึงสามารถประมาณอัตราการเต้นของหัวใจได้



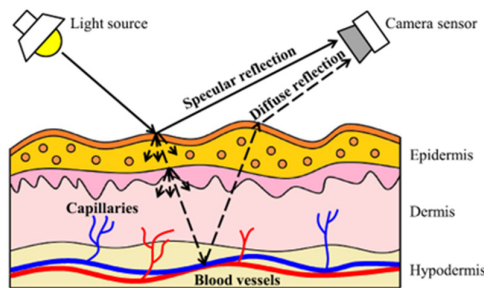
รูปที่ 1 การทำงานของ PPG

ที่มา : <https://www.semanticscholar.org/paper/Reflectance-Based-Organic-Pulse-Meter-Sensor-for-of-Elsamnah-Bilgaiyan/8ee337dd42cb5169b99afbbbedfaf36dcf7863701>

2.2 Remote Photoplethysmography (rPPG)

rPPG เป็นการตรวจจับอัตราการเต้นของหัวใจแบบระยะไกลโดยใช้หลักการคล้ายกับ PPG แต่เป็นการวัดการเปลี่ยนแปลงของการสะท้อนแสงสีแดง สีเขียว และสีน้ำเงินจากผิวหนัง โดยการสะท้อนแสงมีสองลักษณะคือ การสะท้อนแสงปกติ (specular reflection) กับ การสะท้อนแสงแบบกระจาย (diffused reflection) ซึ่งมีความแตกต่างกันดังนี้ การสะท้อนแสงปกติ (specular reflection) เป็นการสะท้อนแสงที่ไม่ถูกดูดซับโดยเนื้อเยื่อผิวหนัง การสะท้อนแบบกระจาย

(diffused reflection) เป็นการสะท้อนแสงที่หลงเหลือจากการดูดซับและการกระเจิงในเนื้อเยื่อผิวหนัง ซึ่งจะแปรผันตามการเปลี่ยนแปลงของปริมาณเลือด



รูปที่ 2 การทำงานของ rPPG

ที่มา : <https://doi.org/10.1109/tbme.2016.2609282>

2.3 Plane-Orthogonal-to-Skin (POS)

งานวิจัย [1] Wang et al. ได้นำเสนอวิธีการกำจัดส่วน specular reflection ออกจากค่าสีแดงเขียว น้ำเงิน ที่ได้จากภาพวิดีโอ โดยได้เขียนไว้ใน Algorithm 1 ของบทความวิจัยดังกล่าว และผู้เขียนได้นำคำสั่ง Python ที่ของอัลกอริทึมนี้ที่นักวิจัยเขียนไว้มาประยุกต์ใช้ โดยตัวแปรที่ใส่ในคำสั่งนี้จะเป็นค่าสีแดงเขียว น้ำเงิน ที่เก็บมาในช่วงเวลาหนึ่งๆ และผลลัพธ์ที่ได้จะเป็นสัญญาณที่คล้ายกับ PPG หรือเรียกว่า rPPG

2.4 การแปลงฟูเรียร์ (Fourier Transform)

การแปลงฟูเรียร์เป็นขบวนการทางคณิตศาสตร์ที่เปลี่ยนฟังก์ชันของเวลาให้อยู่ในรูปของฟังก์ชันของความถี่ หรือเป็นการบอกว่าข้อมูลในช่วงเวลาหนึ่งมีความถี่อะไรบ้าง โดยการแปลงนี้จะมีการคำนวณที่ยุ่งยาก แต่ในโครงงานนี้ผู้เขียนไม่มีพื้นฐานทางคณิตศาสตร์ ผู้เขียนใช้คำสั่ง Python โดยใช้คำสั่ง `scipy.fft.fft()` ที่มีอยู่แล้วใน package Scipy โดยสัญญาณ rPPG ที่ได้จาก POS จะถูกนำมาหาว่ามีความถี่ใดอยู่บ้างด้วยคำสั่งนี้ และจะแปลงความถี่นั้นเป็นอัตราการเต้นของหัวใจ

2.5 การกรองความถี่ในช่วงที่สนใจ (Bandpass Filter)

เนื่องจากอัตราการเต้นของหัวใจของคนจะอยู่ในช่วงความถี่ 0.5-4Hz หรือ 30-240BPM เพื่อให้การหาค่าความถี่จากสัญญาณ rPPG ด้วย `scipy.fft.fft()` ได้ง่ายขึ้นจึงกรองความถี่ในช่วงที่สนใจเอาไว้เท่านั้น การกรองสัญญาณเป็นขบวนการทางคณิตศาสตร์ที่ยุ่งยากแต่มีการสร้างคำสั่ง Python ไว้แล้ว ผู้เขียนเพียงใช้คำสั่ง `scipy.signal.butter()` เพื่อสร้างตัวกรอง และ `scipy.signal.sosfiltfilt()` เพื่อเอาสัญญาณไปผ่านตัวกรอง การที่จะกรองสัญญาณได้ที่ 4Hz ข้อมูลที่ได้จากการประมวลผลจะต้องมีเร็วเป็นอย่างน้อย 2 เท่าซึ่งเป็นทฤษฎีที่ผู้เขียนยังไม่ได้ศึกษาดังนั้นการประมวลผลวิดีโอจะต้องมีความเร็วอย่างน้อย 8 FPS เนื่องจากความเร็วในการประมวลผลของ Raspberry Pi มีค่าประมาณ 5FPS ซ้ำกว่าค่าที่ต้องการจึงไม่สามารถนำมาใช้ได้

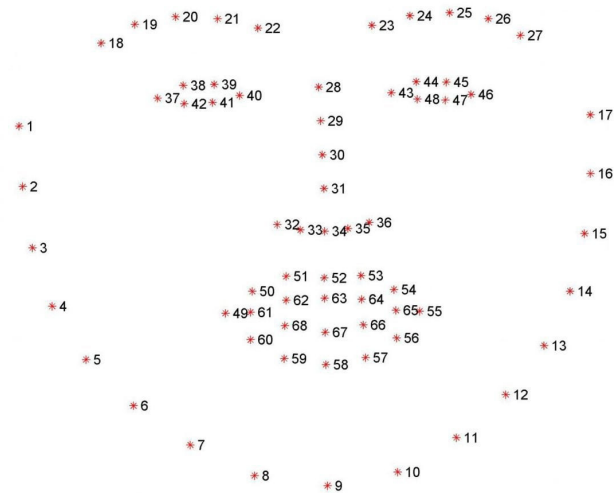
2.6 อัตราการเต้นของหัวใจ

เป็นความเร็วของการบีบตัวของหัวใจในช่วงระยะเวลาหนึ่งโดยทั่วไปนิยมใช้หน่วย "ครั้งต่อนาที" หรือ BPM อัตราหัวใจเต้นสามารถเปลี่ยนแปลงได้ขึ้นกับสรีรวิทยาของร่างกาย เช่นความต้องการออกซิเจนและการขับคาร์บอนไดออกไซด์ของร่างกาย สิ่งที่มีผลกับอัตราหัวใจเต้นได้แก่กิจกรรมของร่างกาย เช่น การออกกำลังกาย การนอนหลับ ความเจ็บป่วย การย่อยอาหาร และยาบางชนิด การแปลงหน่วยจาก Hz หรือครั้งต่อวินาที เป็น BPM สามารถทำได้ง่าย ๆ ตามสมการด้านล่าง

$$BPM = Hz \times 60$$

2.7 การติดตามบริเวณของใบหน้า

เนื่องจากการประมาณอัตราการเต้นของหัวใจต้องใช้สีของใบหน้าที่เราได้จากบริเวณเดียวกันในทุกๆช่วงเวลาจึงต้องมีการติดตามบริเวณที่สนใจ (Region of interest: ROI) ของหน้า โดยมีคำสั่ง Python ในการติดตามจุดสำคัญ (landmark) บนใบหน้าอยู่หลายคำสั่ง dlib เป็นหนึ่งคำสั่งที่ใช้ทำงานง่าย จึงได้เลือกใช้ในโครงงานนี้ โดย dlib มีจุดสำคัญทั้งหมด 68 จุด แบบเรียลไทม์



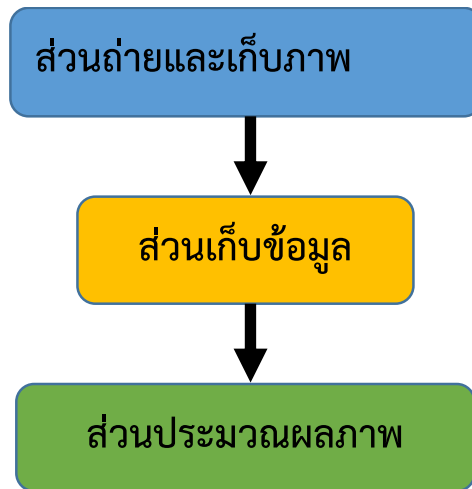
รูปที่ 3 การตรวจจับใบหน้าโดยใช้ dlib

ที่มา : <https://pyimagesearch.com/2017/04/03/facial-landmarks-dlib-opencv-python/>

3 การออกแบบโปรแกรม

3.1 ภาพรวมของโปรแกรม

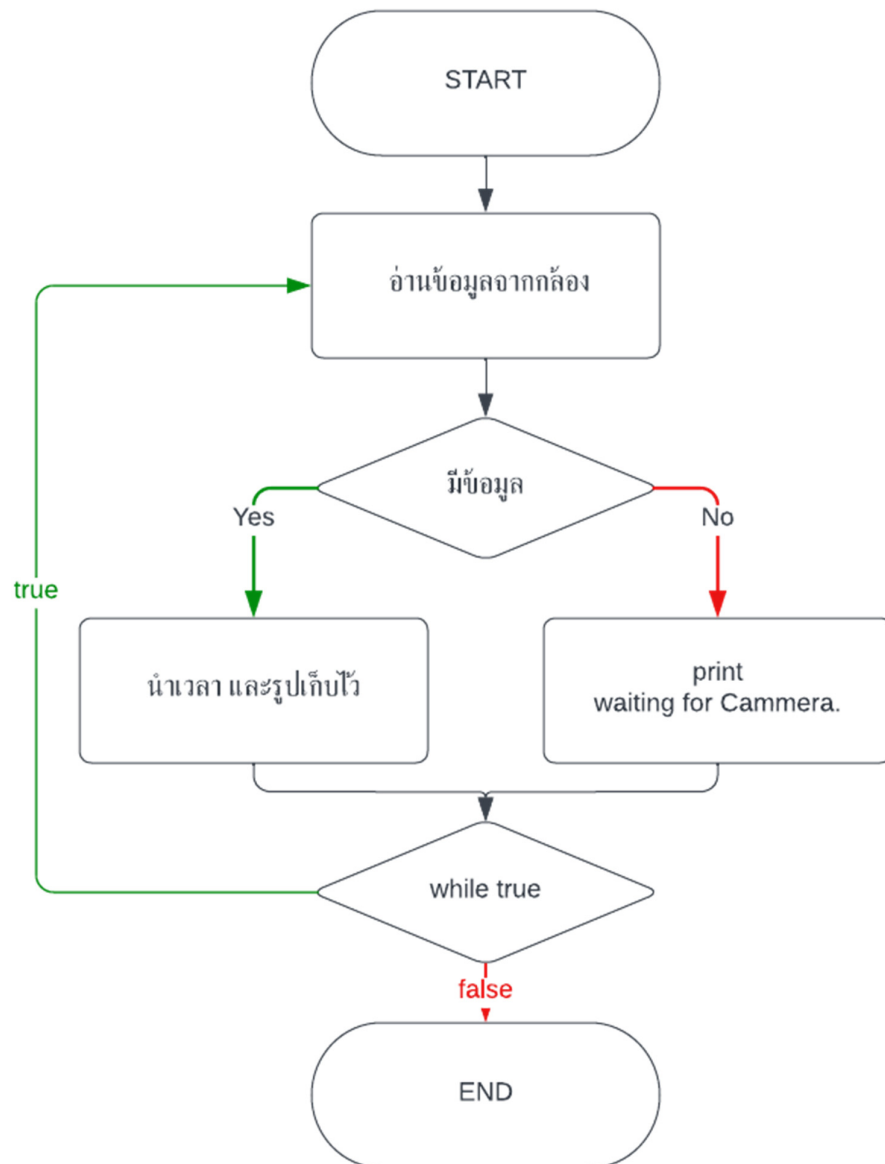
โปรแกรมจะแบ่งออกเป็น 3 ส่วนได้แก่ ส่วนถ่ายและเก็บภาพ ส่วนเก็บข้อมูล และส่วนประมวลผลภาพ ซึ่งส่วนถ่ายภาพ และส่วนเก็บภาพจะนำภาพ และเวลาที่ภาพนั้นถูกถ่ายไปเก็บไว้เพื่อให้ส่วนประมวลผลภาพนำข้อมูลประมวลผลได้ถูกต้อง



รูปที่ 4 ภาพรวมของโปรแกรม

3.2 ส่วนถ่ายและเก็บภาพ

ในการประมวลผลภาพ เราจะต้องตรวจหาใบหน้าของคนที่เราต้องการวัดอัตราการเต้นของหัวใจ โดยใช้ Python module dlib ซึ่งใช้เวลามากกว่าความเร็วในการถ่ายภาพของกล้อง และเนื่องจากเมื่อกล้องถ่ายภาพ ภาพที่ได้จะถูกนำไปเก็บไว้ในหน่วยความจำ buffer ของกล้อง ถ้าเราอ่านข้อมูลภาพออกมาจากหน่วยความจำ buffer ช้ากว่าความเร็วในการถ่ายภาพของกล้อง เราอาจได้ภาพที่ไม่ตรงกับเวลาที่เรากำลังอ่าน ข้อมูลภาพและเวลาของภาพนั้นเป็นสิ่งสำคัญที่เราจะนำไปใช้ในการประมวลผลอัตราการเต้นหัวใจ เราจึงต้องมั่นใจว่าเวลาที่เรากดปุ่มถ่ายภาพเป็นเวลาที่จริง ๆ เราจึงต้องอ่านหน่วยความจำ buffer ของกล้องตลอดเวลาเพื่อให้ไม่มีภาพตกค้างอยู่ในหน่วยความจำ buffer ของกล้อง แล้วนำภาพและเวลาของภาพนั้นๆ ไปเก็บไว้ในตัวแปรตัวหนึ่งโดยเขียนทับตัวแปรนั้นไปเรื่อย เราจะอ่านข้อมูลภาพและเวลาจากตัวแปรนั้น

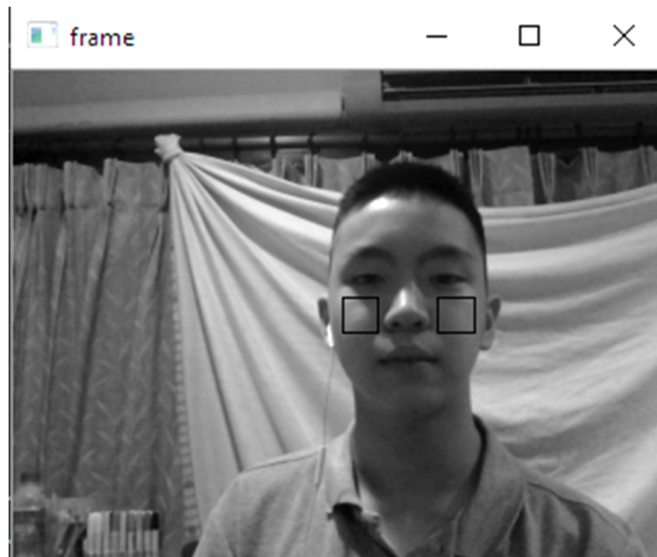


รูปที่ 5 ส่วนถ่ายและเก็บภาพ

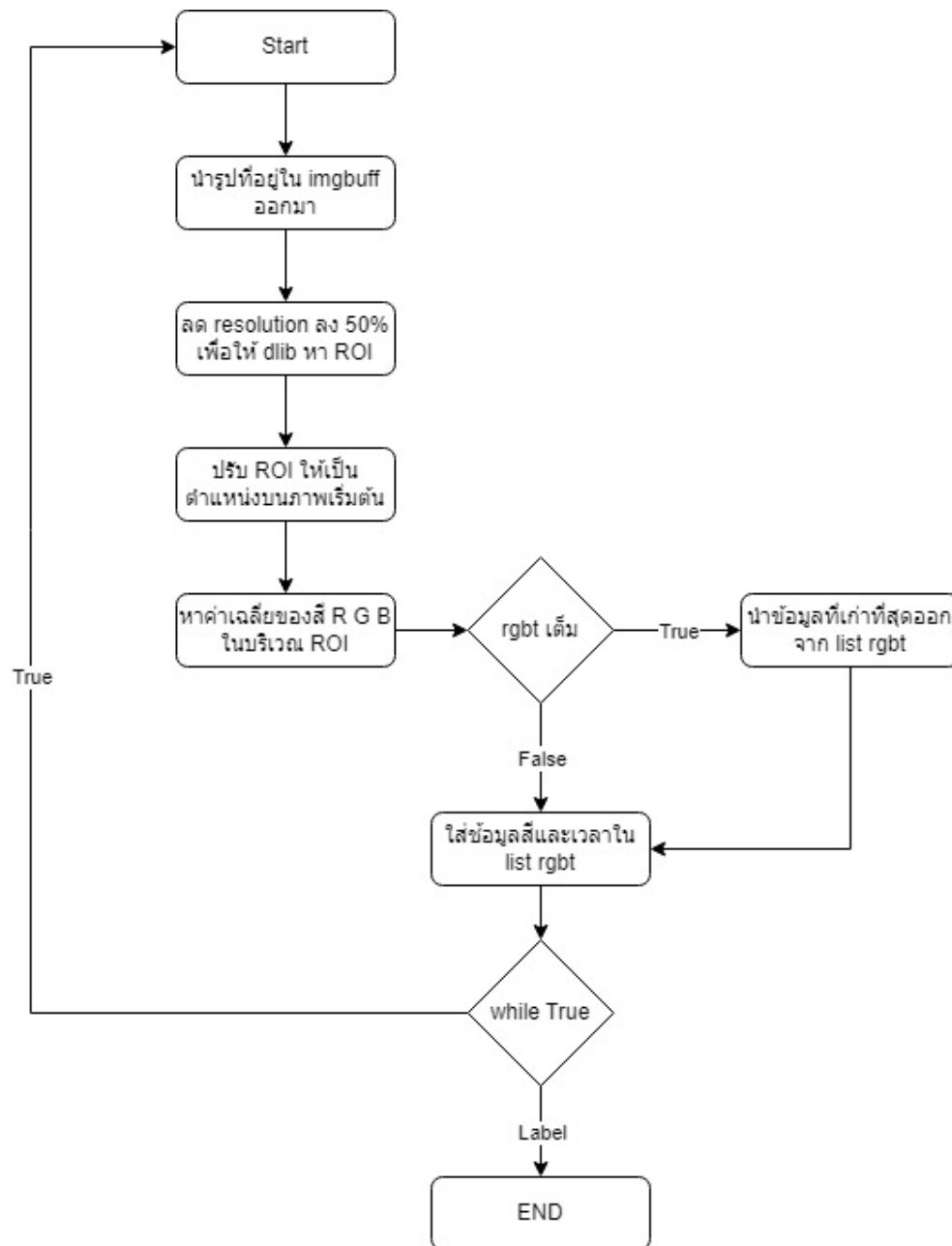
3.3 ส่วนเก็บข้อมูล

ในส่วนนี้เราจะอ่านภาพและเวลาของภาพจากตัวแปรที่จากส่วนที่ 1 แล้วทำการตรวจจับบริเวณ ส่วนของหน้าที่จะใช้เป็นส่วนอ้างอิง ซึ่งการทดลองนี้ใช้ส่วนบริเวณแก้มซ้ายและขวาตามภาพที่ 6 โดยบริเวณแก้มจะกำหนดด้วยจุดสำคัญที่ได้จาก dlib ถ้าภาพที่อ่านมามีขนาดใหญ่ dlib จะใช้เวลานานในการหาจุดสำคัญบนใบหน้า เราจึงลดความละเอียดหรือขนาดของภาพลง 50% เมื่อได้จุดสำคัญแล้วเราจะสเกลจุดสำคัญให้เป็นตำแหน่งพิกเซลของภาพที่ไม่ได้ลดขนาดเพื่อให้ได้ข้อมูลที่เรต้องการคือค่าเฉลี่ยของสีแดง เขียว และน้ำเงิน ในบริเวณนั้นเพื่อใช้เป็นตัวแทนของสัญญาณสี

ของหน้าที่เวลานั้นๆ โดยเราต้องการความยาวของสัญญาณสี 100 จุด ซึ่งจะได้ข้อมูลประมาณ 10 วินาทีถ้าความเร็วในการประมวลผลเป็น 10FPS เราจึงสร้างตัวแปรแบบ array ที่สามารถเก็บข้อมูลสี แดง เขียว น้ำเงิน และเวลา ได้ 100 ชุด และเราจะอ่านภาพและหาข้อมูลสีของภาพแล้วนำข้อมูลมาใส่ใน array ต่อกันไปเรื่อยๆ เมื่อ array เต็มแล้วเราจะเอาข้อมูลที่เก่าที่สุดหรือข้อมูลที่อยู่ด้านซ้ายสุดออกจาก array แล้วเลื่อนข้อมูลทั้งหมดไปทางซ้ายทำให้เหลือช่องใส่ข้อมูล 1 ช่อง และจะทำเช่นนี้ไปเรื่อยๆ การทำแบบนี้สามารถทำได้ง่ายๆโดยใช้ data type แบบ deque ของ Python การ popleft() ข้อมูลของ deque จะย้ายไปทางซ้ายเองแบบอัตโนมัติ



รูปที่ 6 การตรวจจับบริเวณส่วนของหน้า

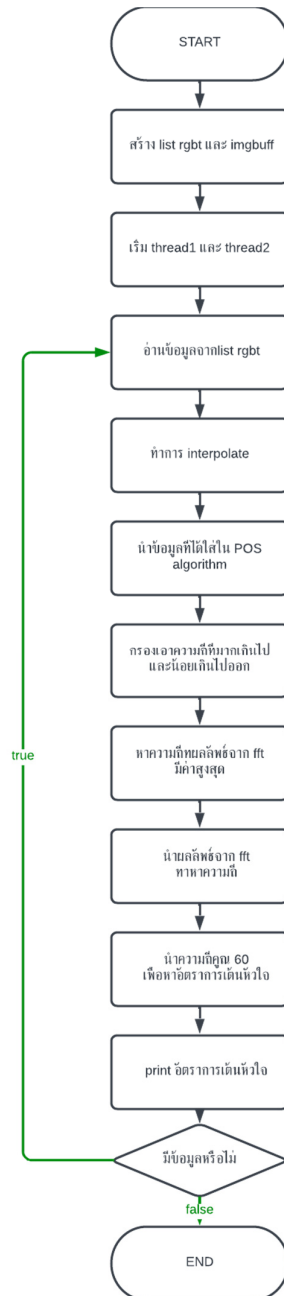


รูปที่ 7 การทำงานของส่วนเก็บข้อมูล

3.4 ส่วนประมวลผลภาพ

เราจะนำตัวแปรที่ ส่วนเก็บข้อมูล ทำไว้ 100 ข้อมูล มาประมวลผลด้วยวิธีการ POS เราจะต้องนำข้อมูลไปทำการ interpolate ก่อนเนื่องจากระยะเวลาที่ใช้ในประมวลผลภาพในแต่ละภาพไม่เท่ากันทำให้ระยะเวลาระหว่างข้อมูลแต่ละข้อมูลไม่เท่ากัน แต่ `scipy.fft.fft()` ต้องข้อมูลที่มี

ระยะเวลาระหว่างแต่ละข้อมูลสี่เท่าๆกัน โดยให้ช่วงเวลาระหว่างข้อมูลเท่ากับค่าเฉลี่ยของ
 ระยะเวลาในการประมวลผลแต่ละภาพ หรือเอาเวลาของข้อมูลสุดท้ายลบกับเวลาของข้อมูลแรก
 หารด้วย 100 (จริงๆต้องหารด้วย 99 แต่ใช้เลข 100 เพื่อให้ช่วงของการ interpolate น้อยกว่า
 ช่วงเวลาของข้อมูลอยู่เล็กน้อย) เราจะ นำข้อมูลที่ได้จากการ interpolate ใส่ใน POS Algorithm
 เพื่อกำจัด specular reflection ให้เหลือเฉพาะสัญญาณ rPPG หลังจากนั้นเราจะนำข้อมูลที่ได้
 จาก POS Algorithm ไปสร้างกราฟความถี่



รูปที่ 8 การสร้างและพัฒนาโปรแกรม

3.5 การนำเข้าคำสั่งที่ต้องใช้

```
import threading
import time
import cv2
import numpy as np
from collections import deque
from scipy import interpolate
import scipy.fft as fft
from scipy.signal import butter, lfilter, freqz, sosfilt, sosfreqz
from scipy.signal import resample, sosfiltfilt, detrend, windows
import matplotlib.pyplot as plt
import dlib
```

threading ใช้เพื่อให้สามารถทำงานหลายอย่างพร้อมกันได้ในที่นี้คือส่วนเก็บภาพ เก็บข้อมูล และประมวลผลภาพ

time ใช้ในการจับเวลา

cv2 ใช้เพื่อถ่ายรูป

numpy , scipy ใช้ในการคำนวณหาอัตราการเต้นหัวใจ

matplotlib ใช้ในการพล็อตกราฟ

dlib ใช้หาบริเวณบนหน้าที่ต้องการ ในที่นี้คือบริเวณแก้ม

3.6 ส่วนเก็บภาพ

```
def thread_function0(imgbuff):
    while True:
        ret, frame = vid.read()
        if ret:
            imgbuff[0] = frame
            imgbuff[1] = int(1000*time.time())-t0
        else:
            print(int(1000*time.time())-t0, ' waiting for Cammera.')
    x0 = threading.Thread(target=thread_function0,args=(imgbuff,), name='thread0', daemon=True)
    x0.start()
```

t0 เป็นเวลาที่เริ่มโปรแกรม การลบค่า t0 ออกจาก time.time() เพื่อให้ค่าเวลาที่เก็บไว้ไม่มีค่ามากเกินไป

3.6.1 เอารูปออกมาจากกล้อง

```
ret, frame = vid.read()
```

3.6.2 ใส่รูปและเวลาเข้าไปในตัวแปร

```
imgbuff[0] = frame  
imgbuff[1] = int(1000*time.time())-t0
```

3.7 ส่วนเก็บข้อมูล

```
def thread_function(imgbuff):  
    while True:  
        frame = imgbuff[0]  
        tt = imgbuff[1]  
        percent = 50  
        img_gray = rescale(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), percent)  
        faces = detector(img_gray)  
        if faces:  
            foundface = 1  
            landmarks = predictor(img_gray, faces[0])  
            landmarks_points = []  
            for n in range(0, 68):  
                x = int((100/percent)*landmarks.part(n).x)  
                y = int((100/percent)*landmarks.part(n).y)  
                landmarks_points.append((x, y))  
            shape=landmarks_points  
  
            ROI1 = frame[shape[29][1]:shape[33][1], #right cheek  
                        shape[54][0]:shape[12][0]]  
  
            ROI2 = frame[shape[29][1]:shape[33][1], #left cheek  
                        shape[4][0]:shape[48][0]]  
            m,n,_ = ROI1.shape  
            tmp = ROI1.reshape((m*n,3))  
            m,n,_ = ROI2.shape  
            tmp = np.concatenate((tmp,ROI2.reshape((m*n,3)))) .mean(axis=0)  
            rgbt.popleft()  
            rgbt.append(np.array([tmp[2],tmp[1],tmp[0],tt]))  
  
            cv2.rectangle(img_gray,(shape[54][0]//2, shape[29][1]//2), #draw rectangle on right and left cheeks  
                          (shape[12][0]//2,shape[33][1]//2), (0,255,0), 0)  
            cv2.rectangle(img_gray, (shape[4][0]//2, shape[29][1]//2),  
                          (shape[48][0]//2,shape[33][1]//2), (0,255,0), 0)  
            cv2.imshow('frame',img_gray)  
            if cv2.waitKey(1) & 0xFF == ord('q'):  
                break  
        else:  
            foundface = 0
```

3.7.1 เอารูปเก็บไว้ออกมา

```
frame = imgbuff[0]  
tt = imgbuff[1]
```

3.7.2 ลดความชัดเจนของรูปและเปลี่ยนรูปเป็น Grayscale เพื่อให้สามารถประมวลผลได้เร็วขึ้น


```
percent = 50
img_gray = rescale(cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY), percent)
```

3.7.3 หาบริเวณบนหน้าที่อยากได้

```
faces = detector(img_gray)
if faces:
    foundface = 1
    landmarks = predictor(img_gray, faces[0])
    landmarks_points = []
    for n in range(0, 68):
        x = int((100/percent)*landmarks.part(n).x)
        y = int((100/percent)*landmarks.part(n).y)
        landmarks_points.append((x, y))
    shape=landmarks_points

    ROI1 = frame[shape[29][1]:shape[33][1], #right cheek
                 shape[54][0]:shape[12][0]]

    ROI2 = frame[shape[29][1]:shape[33][1], #left cheek
                 shape[4][0]:shape[48][0]]
```

3.7.4 หาค่าเฉลี่ยของแต่ละค่าสีในบริเวณที่หาไว้

```
m,n,_ = ROI1.shape
tmp = ROI1.reshape((m*n,3))
m,n,_ = ROI2.shape
tmp = np.concatenate((tmp,ROI2.reshape((m*n,3)))) .mean(axis=0)
```

3.7.5 นำค่าเฉลี่ยและเวลาใส่ในตัวแปร

```
rgbt.popleft()
rgbt.append(np.array([tmp[2],tmp[1],tmp[0],tt]))
```

3.8 ส่วนประมวลผลภาพ

```
while True:
    tmp = rgbt.copy()
    tmp = np.array(tmp)
    r = tmp[:,0]
    g = tmp[:,1]
    b = tmp[:,2]
    tt = tmp[:,3]
    ttt = tt - tt.min()
    rnew = newcolor(r,ttt)
    gnew = newcolor(g,ttt)
    bnew = newcolor(b,ttt)
    fs = 1e3/(ttt.max()/NN)
    print('Sampling Frequency = ', fs)
    data1 = POS(rnew,gnew,bnew,NN,fs)
    data2 = bandpass_filter(data1,0.5,min(0.99*fs/2,4),fs)
    data = bandpass_filter(np.hamming(NN)*data1,0.5,min(0.99*fs/2,4),fs)
    N = int(fs*60)

    dataf = fft.fft(data, n = N)
    dataf1 = fft.fft(data1, n = N)
    dataf2 = fft.fft(data2, n = N)
    f = np.array(range(0,N))*fs/N
    ppgbuff[:600] = resample(data2,600)
    ppgbuffnew[0] = 1

    print('Heart Beat = ',f[abs(dataf[:N//2]).argmax()]*60,' BPM')
    time.sleep(1)
    ii = ii+1
```

3.8.1 เอาข้อมูลที่เก็บไว้ออกมา

```
tmp = rgbt.copy()
tmp = np.array(tmp)
r = tmp[:,0]
g = tmp[:,1]
b = tmp[:,2]
tt = tmp[:,3]
ttt = tt - tt.min()
```

3.8.2 นำข้อมูลเอาออกมาไปทำการ Interpolate

```
rnew = newcolor(r,ttt)
gnew = newcolor(g,ttt)
bnew = newcolor(b,ttt)
fs = 1e3/(ttt.max()//NN)
print('Sampling Frequency = ', fs)
```

```
def newcolor(color,t):
    tmp = np.array(color)
    flinear = interpolate.interpld(t, tmp)
    tnew = (ttt.max()//NN)*np.array(range(NN), dtype = np.int64)
    return flinear(tnew)
```

3.8.3 นำข้อมูลที่ทำทำการ Interpolate แล้วเข้าไปใน POS Algorithm

```
datal = POS(rnew,gnew,bnew,NN,fs)
```

```
def POS(red, green, blue, frame, fs):
    win_size = int(fs*32/20)
    H = np.zeros(frame)
    idx = 0
    while True:
        if idx+win_size > frame:
            break
        R_interval_norm = red[idx:idx+win_size]/red[idx:idx+win_size].mean()
        G_interval_norm = green[idx:idx+win_size]/green[idx:idx+win_size].mean()
        B_interval_norm = blue[idx:idx+win_size]/blue[idx:idx+win_size].mean()
        color_arr = np.array((R_interval_norm, G_interval_norm, B_interval_norm))
        S_1 = (-0.168*R_interval_norm) - (0.331*G_interval_norm) + (0.499*B_interval_norm)
        S_2 = (0.499*R_interval_norm) - (0.418*G_interval_norm) - (0.081*B_interval_norm)
        alpha = S_1.std()/S_2.std()
        h = S_1 + (alpha * S_2)
        H[idx:idx+win_size] = H[idx:idx+win_size] + (h - h.mean())
        idx = idx + 1
    return H
```

3.8.4 นำข้อมูลไปผ่าน filter เพื่อตัดความถี่ที่เป็นไม่ได้ออก

```
datal = POS(rnew,gnew,bnew,NN,fs)
data2 = bandpass_filter(datal,0.5,min(0.99*fs/2,4),fs)
data = bandpass_filter(np.hamming(NN)*datal,0.5,min(0.99*fs/2,4),fs)
```

```
def bandpass_filter(data, lowcut, highcut, fs):
    nyq = 0.5 * fs
    low = float(lowcut) / float(nyq)
    high = float(highcut) / float(nyq)
    order = 3.0
    sos = butter(order, [low, high], btype='band',output='sos')
    bandpass = sosfiltfilt(sos, data)
    return bandpass
```

3.8.5 นำข้อมูลที่ผ่าน filter แล้วมาทำ Fast Fourier transform

```
N = int(fs*60)
dataf = fft.fft(data, n = N)
dataf1 = fft.fft(data1, n = N)
dataf2 = fft.fft(data2, n = N)
```

3.8.6 แสดงผล

```
print('Heart Beat = ', f[abs(dataf[:N//2]).argmax()]*60, ' BPM')
```

4 การทดลองและผลการทดลอง

4.1 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 25 เซนติเมตร

ความเร็วของกล้อง 12 fps

ความสว่าง 135 lux

	อัตราการเต้นของ หัวใจที่วัดได้	อัตราการเต้นของ หัวใจจริง	คลาดเคลื่อน
ทดลองครั้งที่ 1	85	86	1
ทดลองครั้งที่ 2	84	84	0
ทดลองครั้งที่ 3	87	89	2

คิดเป็นอัตราการคลาดเคลื่อน 1.12%

4.2 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 50 เซนติเมตร

ความเร็วของกล้อง 12 fps

ความสว่าง 139 lux

	อัตราการเต้นของ หัวใจที่วัดได้	อัตราการเต้นของ หัวใจจริง	คลาดเคลื่อน
ทดลองครั้งที่ 1	83	86	3
ทดลองครั้งที่ 2	81	85	4
ทดลองครั้งที่ 3	93	89	5

คิดเป็นอัตราการคลาดเคลื่อน 4.62%

4.3 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 75 เซนติเมตร

ความเร็วของกล้อง 13 fps

ความสว่าง 136 lux

	อัตราการเต้นของ หัวใจที่วัดได้	อัตราการเต้นของ หัวใจจริง	คลาดเคลื่อน
ทดลองครั้งที่ 1	74	81	7
ทดลองครั้งที่ 2	77	83	6
ทดลองครั้งที่ 3	84	78	6

คิดเป็นอัตราการคลาดเคลื่อน 7.85%

4.4 ทดลองวัดอัตราการเต้นของหัวใจจากระยะทาง 100 เซนติเมตร

ความเร็วของกล้อง 12 fps

ความสว่าง 137 lux

	อัตราการเต้นของ หัวใจที่วัดได้	อัตราการเต้นของ หัวใจจริง	คลาดเคลื่อน
ทดลองครั้งที่ 1	81	73	8
ทดลองครั้งที่ 2	66	70	4
ทดลองครั้งที่ 3	83	71	12

คิดเป็นอัตราการคลาดเคลื่อน 11.21%

4.5 ทดลองวัดอัตราการเต้นของหัวใจโดยใช้ภาพถ่าย

จากการทดลองโปรแกรมแสดงผลมั่วๆไม่หยุดอยู่ที่ค่าใดค่าหนึ่ง

5 สรุปโครงการและข้อเสนอแนะ

5.1 สรุปผลโครงการ

จากการทดลองพบว่า โปรแกรมที่ได้พัฒนาขึ้นมาสามารถนำมาใช้งานได้จริง โดยยังอยู่ใกล้กล้อง
ยังมีความคลาดเคลื่อนน้อย แต่ยังไม่สามารถทำให้ไม่มีการคลาดเคลื่อนเลยได้ เนื่องจากมีตัวแปรที่
อยู่เหนือการควบคุมของโปรแกรมเช่น ความชัดของกล้อง ความสว่างของแสง ทิศทางที่แสงส่อง
มา ความเร็วของเครื่องมือที่ใช้ประมวลผล เป็นต้น

5.2 ข้อเสนอแนะ

เนื่องจากเวลาส่วนมากถูกใช้ในการตรวจจับบริเวณส่วนของใบหน้า ที่จะใช้เป็นส่วนอ้างอิง
จึงคิดว่าต้องหาวิธีที่ใช้ในการตรวจจับ ใบ หน้า แบบ อื่นๆที่ใช้เวลาน้อยลง หรือเปลี่ยนจากบริเวณ
หน้าเป็นบริเวณอื่นของร่างกายเช่น ฝ่ามือ แขน ปลายนิ้ว เป็นต้น เพื่อให้ไม่ต้องมีการตรวจจับ
อะไรเลย

6 บรรณานุกรม

[1] W. Wang, A. C. den Brinker, S. Stuijk and G. de Haan, "Algorithmic Principles of Remote PPG," in IEEE Transactions on Biomedical Engineering, vol. 64, no. 7, pp. 1479-1491, July 2017, doi: 10.1109/TBME.2016.2609282.