1 การวางแผนการเคลื่อนที่

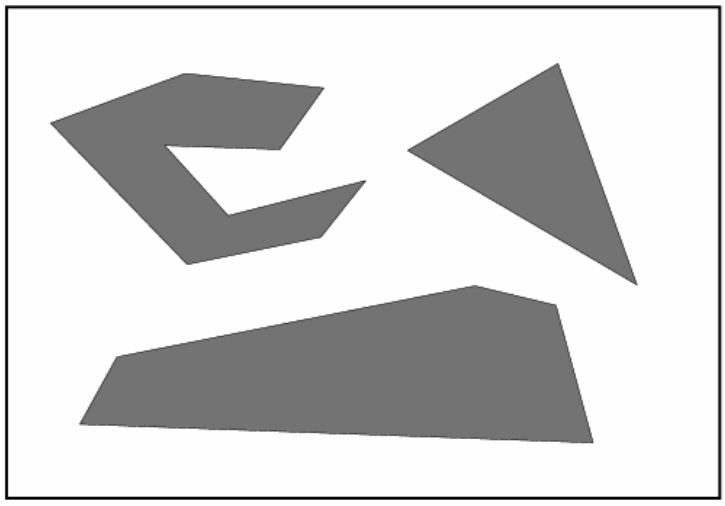
“ถ้าขยับไม่ได้ ก็ไม่ใช่หุ่นยนต์” ความสามารถในการเคลื่อนที่นับเป็นคุณสมบัติที่สำคัญอย่างหนึ่งของหุ่นยนต์ จนมีบางคนให้คำนิยามว่าหุ่นยนต์ที่จริงแล้วก็คือคอมพิวเตอร์บวกกับอุปกรณ์ทางกลที่ทำให้มันเคลื่อนที่ได้ การทำให้หุ่นยนต์เคลื่อนที่ได้ไม่ใช่เรื่องยาก แต่จะให้เคลื่อนที่อย่างมีประโยชน์และมีประสิทธิภาพแบบอัตโนมัตินั้นไม่ใช่เรื่องง่ายเลย การที่จะให้หุ่นยนต์ไปทำงานตามที่ต้องการเช่นไปเคลื่อนย้ายวัตถุสิ่งของที่กำหนด มันจะต้องเคลื่อนตัวเข้าไปอยู่ในบริเวณที่สามารถทำงานนั้นได้ก่อน การเคลื่อนจากที่หนึ่งไปยังอีกที่หนึ่ง มีข้อต้องพิจารณาหลายประการด้วยกัน เช่นการเลือกเส้นทางที่เหมาะสมเพื่อที่หุ่นยนต์สามารถไปถึงจุดหมายโดยไม่ชนกับสิ่งกีดขวาง การสั่งงานระบบขับเคลื่อนให้หุ่นยนต์เคลื่อนไปตามเส้นทางที่ต้องการเช่นควบคุมล้อให้หมุนอย่างเหมาะสม การปรับเส้นทางขณะที่หุ่นยนต์กำลังเดินทางให้สอดคล้องกับสิ่งแวดล้อมที่เปลี่ยนแปลงเช่น สิ่งกีดขวางที่เคลื่อนไหวได้ หรือหุ่นยนต์ตัวอื่นๆ ที่ทำงานในบริเวณเดียวกัน ฯลฯ อันที่จริงกิจกรรมต่างๆ ที่อาจฟังไม่คุ้นหูเหล่านี้ล้วนเป็นสิ่งที่เราคุ้นเคยดี มนุษย์และสัตว์มีความสามารถในการเคลื่อนไหวหลบหลีกสิ่งกีดขวาง ทั้งที่อยู่นิ่งหรือเคลื่อนไหวได้อย่างคล่องแคล่ว ซึ่งเราทำไปโดยอัตโนมัติด้วยความง่ายดาย แต่เราก็ไม่สามารถอธิบายแจกแจงได้ว่าเราทำกิจกรรมเหล่านี้ได้อย่างไร มีขั้นตอนอย่างไรบ้าง มนุษย์มีความพยายามที่จะสร้างสิ่งประดิษฐ์ให้มีความสามารถในการเคลื่อนที่แบบอัตโนมัติมานานแล้ว แต่ถึงในปัจจุบันก็ยังมีขีดความสามารถจำกัดเมื่อเปรียบเทียบกับสิ่งมีชีวิต ในบทนี้เราจะขอกล่าวถึงความรู้พื้นฐาน ที่มีจุดมุ่งหมายจะทำให้หุ่นยนต์มีความสามารถดังกล่าวได้ เราจะพิจารณาปัญหาการวางแผนการเคลื่อนที่ของหุ่นยนต์ การวางแผนการเคลื่อนที่ของหุ่นยนต์ก็คือการกำหนดว่าหุ่นยนต์จะต้องเคลื่อนที่อย่างไรเพื่อที่จะเดินทางจากจุดเริ่มต้นไปยังจุดหมายที่กำหนดได้ โดยไม่ชนกับสิ่งกีดขวางใดๆ ที่แวดล้อมมันอยู่ ปัญหาการวางแผนการเคลื่อนที่นี้มีความชัดเจนสามารถทำความเข้าใจได้ง่าย และแนวความคิดในการวิเคราะห์ปัญหานี้ก็เป็นพื้นฐานทางทฤษฎีของงานอื่นๆเกี่ยวกับหุ่นยนต์อีกหลายชิ้น ในบทนี้เราจะเริ่มจากการวางแผนการเคลื่อนที่กรณีง่ายสุด นั่นคือเมื่อหุ่นยนต์อยู่ในรูปของจุด จากนั้นเราจะแสดงให้เห็นว่าปัญหาการวางแผนการเคลื่อนที่แบบซับซ้อนขึ้นก็สามารถลดทอนให้เป็นปัญหาการวางแผนการเคลื่อนที่ของจุดภายใต้ข้อจำกัดบางประการ ซึ่งข้อจำกัดเหล่านี้จะค่อยๆ ถูกตัดออกไปเมื่อเราย้อนกลับมาพิจารณาปัญหาการวางแผนการเคลื่อนที่อีกครั้งในบทท้ายๆ

1.1 การวางแผนการเคลื่อนที่ของหุ่นยนต์จุด

โดยทั่วไปหุ่นยนต์มีลักษณะที่หลากหลาย บางตัวอาจมีล้อในการขับเคลื่อน บางตัวอาจมีขา หรือบางตัวอาจประกอบไปด้วยข้อต่อต่างๆ ที่ช่วยในการเคลื่อนไหว เพื่อความสะดวกในการวิเคราะห์ในบทนี้เราจะขอสมมุติให้หุ่นยนต์เป็นวัตถุแข็งเกร็ง (rigid body) ที่สามารถเคลื่อนที่ได้อย่างอิสระในสิ่งแวดล้อมที่มีสิ่งกีดขวางเป็นวัตถุแข็งเกร็งเช่นกัน เราจะยังไม่กังวลว่าหุ่นยนต์ของเราจะเคลื่อนที่ได้อย่างไรในบทนี้

ลองจินตนาการว่าเรามีห้องรูปสี่เหลี่ยมที่ไม่มีประตู มีหุ่นยนต์ขนาดเล็กมากจนสามารถพิจารณาว่ามันเป็นจุดได้ หุ่นยนต์ตัวนี้สามารถเคลื่อนที่อย่างอิสระบนพื้นห้อง และบนพื้นห้องนี้ก็มีสิ่งของเป็นรูปหลายเหลี่ยมวางกระจายอยู่ทั่วไป โดยหุ่นยนต์ไม่สามารถเดินข้ามสิ่งของเหล่านี้และมันก็ไม่มีแรงพอที่จะผลักให้สิ่งของที่กีดขวางอยู่เคลื่อนที่ได้ ปัญหามีอยู่ว่าถ้าเราต้องการให้หุ่นยนต์เคลื่อนที่จากจุดเริ่มต้นไปยังจุดหมายที่ต้องการโดยไม่ชนกับสิ่งของที่วางอยู่เลย เราจะทำอย่างไรดี อยากให้ผู้อ่านลองคิดวิธีของตัวเองก่อนที่จะดูตัวอย่างอัลกอริทึมสำหรับแก้ปัญหานี้

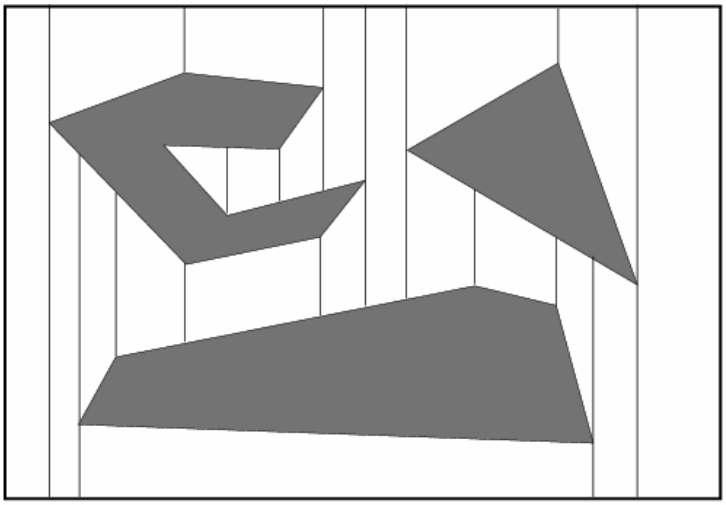
เชื่อว่าหลายคนต้องนึกถึงการใช้กราฟเข้ามาช่วยในการวางแผนเส้นทาง อัลกอริทึมที่จะนำเสนอนี้ก็ใช้กราฟเป็นอุปกรณ์หลักในการค้นหาเส้นทาง หลักการโดยคร่าวๆ ก็คือเราจะแบ่งพื้นห้องส่วนที่ไม่มีสิ่งของวางทับอยู่ออกเป็นส่วนๆ จากนั้นเราจะสร้างกราฟแทนการอยู่ติดกันของส่วนย่อยเหล่านี้ การคำนวณหาเส้นทางจึงทำได้โดยการค้นในกราฟ ว่ามีทางเดินจากส่วนย่อยเริ่มต้นที่คลุมตำแหน่งเริ่มต้น ไปส่วนย่อยที่ต้องการหรือไม่ รูปที่ 1.1 แสดงพื้นห้องล้อมรอบด้วยกำแพงทึบและมีสิ่งกีดขวางรูปหลายเหลี่ยมวางกระจายอยู่ เราจะแสดงขั้นตอนโดยละเอียดของอัลกอริทึมนี้และแสดงให้ดูว่ามีอะไรเกิดขึ้นกับตัวอย่างในรูปที่ 1.1 ในแต่ละขั้นตอน



รูปที่ 1.1 workspace ที่ถูกล้อมรั้ว

<u>ขั้นที่หนึ่ง</u>

สร้างแผนผังสี่เหลี่ยมคางหมู (trapezoidal map) ของบริเวณอิสระด้วยการหั่นพื้นห้องที่ไม่ถูกทับด้วยสิ่งกีดขวางตามแนวตั้งออกเป็นสี่เหลี่ยมคางหมู โดยใช้พิกัดตามแกน *x* ของจุดยอดของสิ่งกีดขวางเป็นตำแหน่งหั่น รูปที่ 1.2 แสดงผลของการแบ่งซึ่งประกอบไปด้วยรูปสี่เหลี่ยมคางหมู 20 รูป และรูปสามเหลี่ยม 1 รูป ที่จริงรูปสามเหลี่ยมรูปนี้ก็เป็นกรณีพิเศษของสี่เหลี่ยมคางหมูที่ลดรูปเป็นสามเหลี่ยมเมื่อหนึ่งในด้านขนานของมันมีขนาดเป็นศูนย์

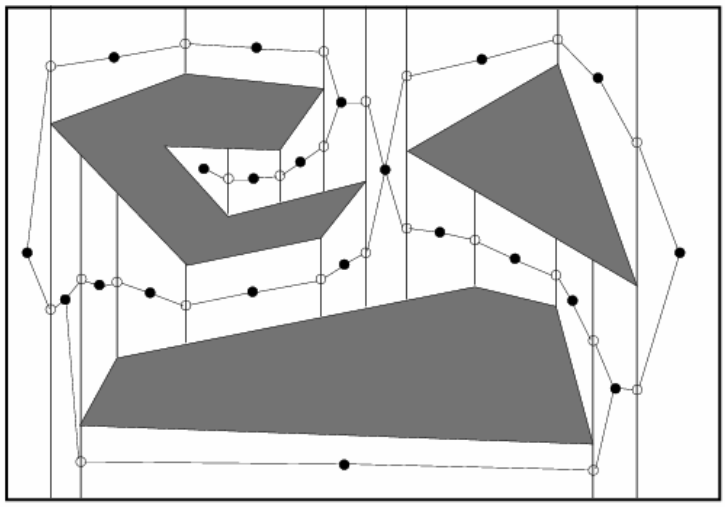


รูปที่ 1.2 การแบ่งบริเวณอิสระเป็นแผนผังสี่เหลี่ยมคางหมู

แผนผังสี่เหลี่ยมคางหมูสามารถหาได้โดยใช้หลักการกวาดด้วยเส้น (line sweep algorithm) โดยใช้พิกัดตามแกน *x* ของจุดยอดของสิ่งกีดขวางเป็นจุดเกิดเหตุ และใช้เวลา $O(n \log n)$ โดยที่ *n* คือจำนวนจุดยอดทั้งหมดของสิ่งกีดขวาง เวลาที่ต้องใช้นี้ถูกกำหนดด้วยเวลาที่ใช้ในการเรียงพิกัดในแกน *x* ของจุดยอดทั้งหมด

<u>ขั้นที่สอง</u>

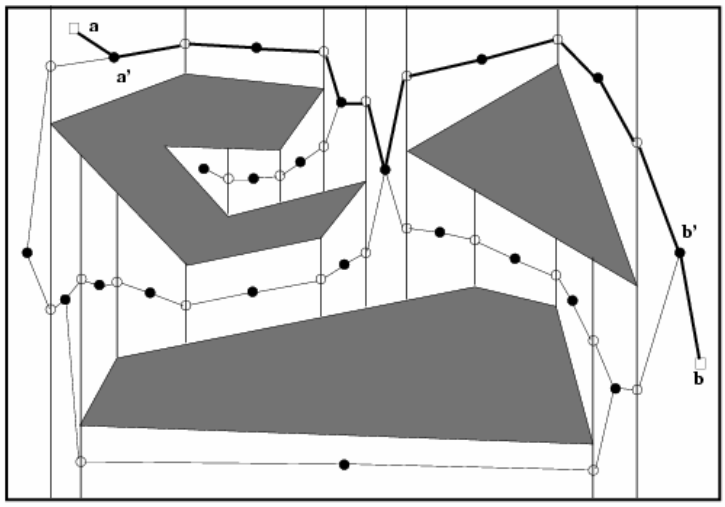
สร้างแผนที่ทางเดิน (road map) จากแผนผังสี่เหลี่ยมคางหมู ซึ่งก็คือกราฟที่บรรยายลักษณะการอยู่ติดกันของสี่เหลี่ยมคางหมูที่อยู่ในแผนผัง โดยเราจะกำหนดให้มีจุดยอดที่จุดกึ่งกลางของสี่เหลี่ยมคางหมูแต่ละรูปในแผนผัง (จุดดำในรูปที่ 1.3) และจุดกึ่งกลางของแต่ละเส้นในแนวตั้ง (จุดขาวในรูปที่ 1.3) จากนั้นเราจะลากเส้นเชื่อมระหว่างจุดกึ่งกลางของสี่เหลี่ยมคางหมูกับจุดกึ่งกลางของเส้นแนวตั้งที่ล้อมมันอยู่ ได้ผลเป็นแผนที่ทางเดินแสดงในรูปที่ 1.3 ถ้าเราใช้โครงสร้างข้อมูลแบบรายการเส้นเชื่อมสองฝั่ง (doubly-connected edge list) ในการแทนแผนผังสี่เหลี่ยมคางหมู เราก็สามารถที่จะหาแผนที่ทางเดินได้ภายในเวลา $O(n)$ ด้วยการแวะตามรายการนี้



รูปที่ 1.3 การสร้างแผนที่ทางเดินจากแผนผังสี่เหลี่ยมคางหมู

ขั้นที่สาม

ใช้การค้น (อาจเป็นการค้นแนวกว้าง หรือแนวลึก หรืออาจนำระยะทางระหว่างจุดยอดมาประกอบการพิจารณา) เพื่อหาเส้นทางเดินจากจุดเริ่มต้นไปยังจุดหมาย โดยขั้นแรกจะต้องหาว่าจุดเริ่มต้นและจุดหมายอยู่ในสี่เหลี่ยมคางหมูใด เมื่อทราบแล้ว จึงทำการค้นหาเส้นทางระหว่างจุดยอดของสี่เหลี่ยมคางหมูทั้งสองในแผนที่ทางเดินที่ได้จากขั้นที่สอง เส้นทางที่ได้นี้นำมาประกอบกับเส้นตรงจากจุดเริ่มต้นไปยังจุดยอดของสี่เหลี่ยมคางหมูเริ่มต้น และเส้นตรงจากจุดยอดของสี่เหลี่ยมคางหมูจุดหมาย รวมกันเป็นเส้นทางจากจุดเริ่มต้นไปยังจุดหมาย ที่ทำเช่นนี้ได้เพราะเราแน่ใจได้ว่าเส้นตรงระหว่างสองจุดใดๆ ในสี่เหลี่ยมคางหมูต้องอยู่ภายในสี่เหลี่ยมคางหมูนั้นอย่างแน่นอนเนื่องจากสี่เหลี่ยมคางหมูเป็นรูปเหลี่ยมคอนเวกซ์ เนื่องจากจำนวนของจุดยอดเหล่านี้แปลผันเชิงเส้นกับจำนวนส่วนของเส้นตรงที่ประกอบเป็นสิ่งกีดขวาง หากใช้การค้นแนวกว้างเราจึงได้ว่าการค้นหาเส้นทางกินเวลา $O(n)$ ในส่วนการหาว่าจุดเริ่มต้นและจุดหมายอยู่ในสี่เหลี่ยมคางหมูใด หากใช้การพิจารณาสี่เหลี่ยมทีละอันจะกินเวลา $O(n)$ (แต่จริงๆ เราทำได้เร็วกว่านี้โดยใช้อัลกอริทึมเชิงสุ่มที่มีค่าคาดหมายของเวลาทำงานเป็น $O(\log n)$) ดังนั้นโดยรวมอัลกอริทึมนี้กินเวลา $O(n \log n)$ รูปที่ 1.4 แสดงตัวอย่างเส้นทางที่หาได้ จากจุดเริ่มต้น a ไปยังจุดหมาย b ที่ประกอบด้วยเส้นตร **aa'** เชื่อมจุดเริ่มต้นกับจุดยอดของสี่เหลี่ยมที่มันอยู่ภายใน กับเส้นตรง **b'b** ที่เชื่อมจุดหมายกับจุดยอดของสี่เหลี่ยมที่มันอยู่ภายใน และเส้นทางจาก **a'** ไป **b'** ที่ได้จากการค้นแผนที่ทางเดิน



รูปที่ 1.4 เส้นทางจากจุดเริ่มต้น **a** ไปยังจุดหมาย **b**

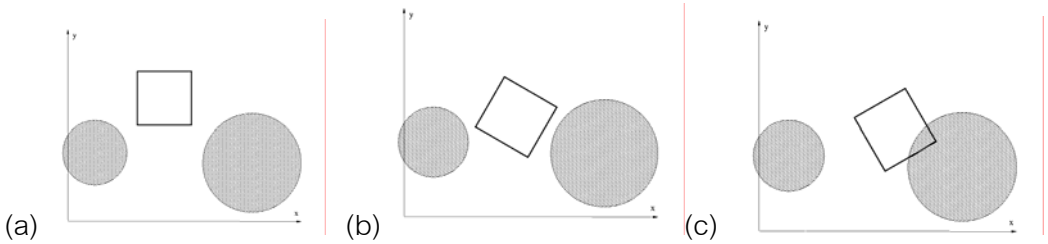
อัลกอริทึมที่ยกให้ดูนี้เป็นตัวอย่างง่ายๆ ภายใต้ข้อจำกัดที่อาจไม่สมจริงนักสำหรับหลายสถานการณ์ ลองคิดดูว่าจะทำอย่างไรถ้าหุ่นยนต์เรามีขนาดใหญ่เกินกว่าที่จะมองให้เป็นจุดได้ กรณีง่ายสุดก็คือในกรณีที่หุ่นยนต์เป็นรูปวงกลม นอกจากนี้หากแทนที่จะวางแผนการเส้นทางบนพื้นห้องกลายเป็นการวางแผนการเคลื่อนที่ของหุ่นยนต์ที่ลอยตัวได้อย่างอิสระในอวกาศ เราจะทำอย่างไรดี ก่อนที่จะไปพิจารณาปัญหาต่างๆ เหล่านี้ ต้องขออธิบายพื้นฐานที่จะช่วยให้เราสามารถพูดถึงปัญหาเหล่านี้ได้สะดวกขึ้น

1.2 บริเวณทำงานและคอนฟิกุเรชัน

เทคโนโลยีหุ่นยนต์ก็เหมือนศาสตร์อื่นๆ ที่มีคำศัพท์เฉพาะทาง ช่วยอำนวยความสะดวกในการกล่าวถึงสิ่งที่ควรเป็นที่รู้จักดีในกลุ่มผู้ศึกษาศาสตร์นั้นๆ ในส่วนนี้เราจะขอแนะนำคำศัพท์ที่เกี่ยวข้องกับการวางแผนการเคลื่อนที่ ซึ่งได้แก่ คำว่า บริเวณทำงาน (workspace), สิ่งกีดขวาง (obstacle), บริเวณต้องห้าม (forbidden space), บริเวณอิสระ (free space), และ คอนฟิกุเรชัน (configuration) โดยต้องขอเรียนไว้ก่อนว่าขณะนี้ยังไม่มีการบัญญัติศัพท์สำหรับเทคโนโลยีหุ่นยนต์โดยเฉพาะ ผมจึงขออนุญาตเลือกคำศัพท์ภาษาไทยมาใช้ตามที่เห็นเหมาะสม โดยบางคำขอใช้ตามศัพท์บัญญัติในสาขาที่เกี่ยวข้องเช่น คณิตศาสตร์ และ คอมพิวเตอร์ คำศัพท์ที่เกี่ยวกับอัลกอริทึมและโครงสร้างข้อมูลส่วนใหญ่จะนำมาจากหนังสืออัลกอริทึมของ รศ. ดร. สมชาย ประสิทธิจูตระกูล

หุ่นยนต์เป็นวัตถุที่มีตัวตน (ในบทนี้เรากำหนดให้มันเป็นวัตถุแข็งเกร็งชิ้นเดียว) มันจึงต้องกินเนื้อที่ เราเรียกอาณาบริเวณในโลกจริงที่หุ่นยนต์อาศัยอยู่ว่า***บริเวณทำงาน*** โดยทั่วไป บริเวณทำงานจะมีสามมิติเหมือนกับโลกจริงที่เรา

รับรู้ได้ แต่ในบางปัญหา บริเวณทำงานอาจถูกกำหนดให้มีเพียงสองมิติ (เช่นในตัวอย่างปัญหาการวางแผนทางเดินของหุ่นยนต์จุดที่ยกให้ดูไปแล้ว) ซึ่งพอเพียงแล้วสำหรับปัญหานั้น และที่สำคัญจะช่วยลดความยุ่งยากในการคำนวณดังที่เราจะได้เห็นชัดเจนขึ้นเมื่อเข้าสู่ส่วนรายละเอียดต่อไป ในบริเวณทำงานหนึ่งๆ นั้น นอกจากหุ่นยนต์แล้วยังเป็นที่อยู่ของวัตถุที่เราจะเรียกว่า***สิ่งกีดขวาง*** สิ่งกีดขวางเป็นวัตถุในบริเวณทำงานที่เราไม่สามารถควบคุมได้ กำหนดในบทนี้ให้เป็นวัตถุแข็งเกร็งที่อยู่กับที่ แต่ในโลกของความเป็นจริง สิ่งกีดขวางอาจเป็นวัตถุยืดหยุ่นก็ได้ อาจเป็นวัตถุที่อยู่ติดกับที่ หรืออาจเป็นวัตถุที่ถูกผลักให้เคลื่อนได้ หรืออาจจะเคลื่อนที่เองได้ทั้งที่เราอาจรู้หรือไม่รู้ลักษณะการเคลื่อนที่ของมันก็ได้ เราจะเรียกพื้นที่ที่มีสิ่งกีดขวางอยู่ว่า***บริเวณต้องห้าม*** และเราจะเรียกพื้นที่ในบริเวณทำงานที่ไม่มีสิ่งกีดขวางอยู่ว่า***บริเวณอิสระ*** ข้อสำคัญคือสิ่งกีดขวางและหุ่นยนต์ไม่สามารถกินที่ในตำแหน่งเดียวกันในบริเวณทำงานได้ ซึ่งที่จริงก็เป็นไปตามธรรมชาติ ที่เนื้อของสิ่งของสองสิ่งนั้นไม่สามารถซ้อนทับในตำแหน่งเดียวกันในเวลาเดียวกันได้ ตอนนี้ลองดูตัวอย่างของบริเวณทำงานที่เป็นสองมิติ และมีสิ่งกีดขวางรูปกลมกับหุ่นยนต์รูปสี่เหลี่ยมจตุรัสอาศัยอยู่ในรูปที่ 1.5



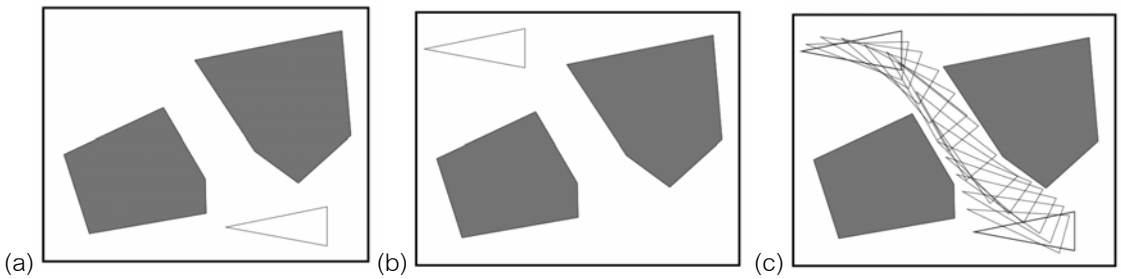
รูปที่ 1.5 workspace ที่มีหุ่นยนต์รูปสี่เหลี่ยมจตุรัสและสิ่งกีดขวางรูปกลม

เราจะเห็นได้ว่าหุ่นยนต์สามารถครอบคลุมเนื้อที่ต่างๆ กันของบริเวณทำงานได้โดยการเปลี่ยน ตำแหน่ง (position) และแนวการวางตัว (orientation) ตำแหน่งและแนวการวางตัวที่กล่าวถึงนี้คือตัวกำหนดที่เรียกว่า ***คอนฟิกุเรชัน*** ของหุ่นยนต์ สังเกตว่าในรูปที่ 1.5(c) หุ่นยนต์อยู่ในคอนฟิกุเรชันที่เป็นไปไม่ได้เพราะมันทับกับสิ่งกีดขวาง

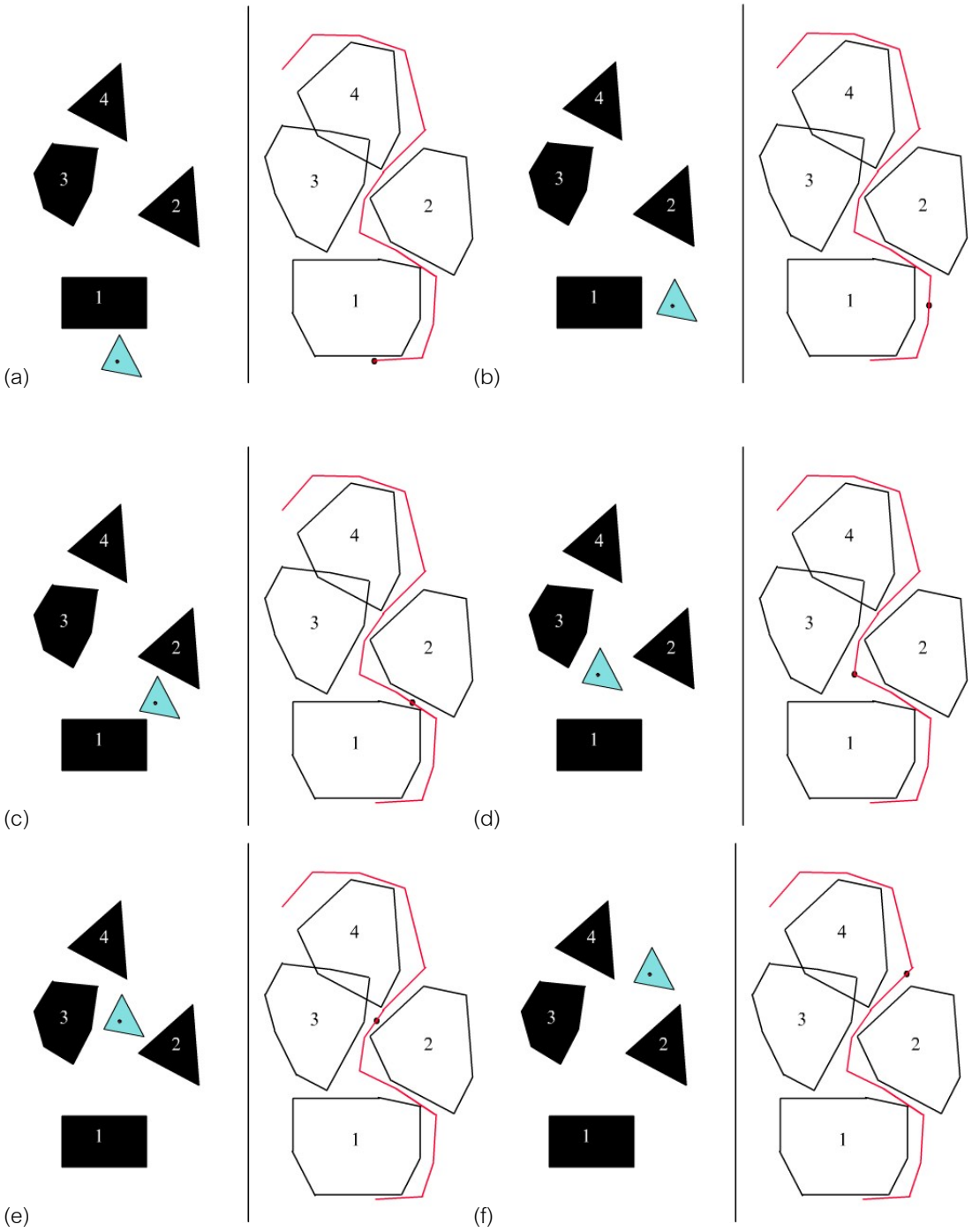
คอนฟิกุเรชันของหุ่นยนต์คือพารามิเตอร์ทั้งหมดซึ่งเมื่อประกอบกับรูปแบบของหุ่นยนต์ตัวนั้น สามารถใช้คำนวณว่าหุ่นครอบคลุมพื้นที่ใดในบริเวณทำงาน ลองพิจารณาหุ่นตัวอย่างที่เป็นรูปสี่เหลี่ยมจตุรัสกว้างหนึ่งหน่วย กำหนดให้หุ่นตัวนี้ไม่สามารถหมุนได้แต่เลื่อนได้โดยที่ขอบทั้งสองของหุ่นขนานกับแกน x และ y ของบริเวณทำงานเสมอ จากรูปแบบของหุ่นที่กำหนดให้ จะเห็นได้ว่าเราสามารถรู้ได้ว่าหุ่นตัวนี้จะกินเนื้อที่อย่างไรเมื่อเรารู้ว่ามุมซ้ายล่างของหุ่นตัวนี้อยู่ที่พิกัดใดในบริเวณทำงาน (กำหนดให้เป็นจุดที่มุมเพื่อความสะดวกในการอ้างอิง จุดอื่นของหุ่นก็ใช้ได้เช่นกัน แต่สำคัญที่ต้องยึดจุดเดียวกันตลอดการวิเคราะห์) ดังนั้นในกรณีนี้พิกัดของมุมซ้ายล่างของหุ่นก็คือคอนฟิกุเรชันของหุ่น (ในกรณีที่หุ่นหมุนได้ นอกเหนือจากพิกัดของจุดหนึ่งบนตัวหุ่น คอนฟิกุเรชันยังต้องประกอบไปด้วยมุมของแนวการวางตัวของหุ่น)

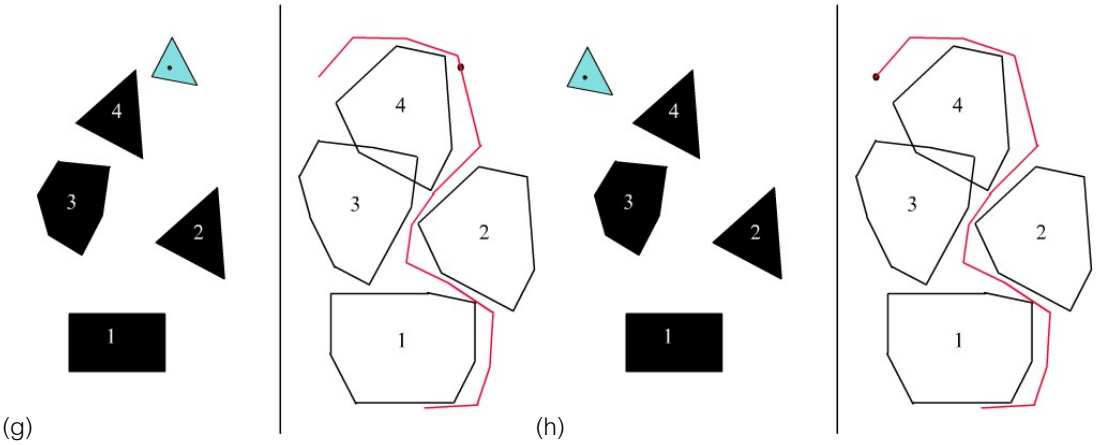
1.3 การวางแผนการเคลื่อนที่คืออะไร

การวางแผนการเคลื่อนที่ ก็คือการกำหนดว่าคอนฟิกุเรชันของหุ่นยนต์จะต้องเปลี่ยนแปลงอย่างไร เพื่อให้หุ่นยนต์เคลื่อนตัวจากคอนฟิกุเรชันเริ่มต้นไปยังคอนฟิกุเรชันที่ต้องการ โดยไม่ชนกับสิ่งกีดขวางเลย ดังในตัวอย่างในรูปที่ 1.6(c) แสดงหุ่นยนต์เคลื่อนตัวจากคอนฟิกุเรชันเริ่มต้น (รูปที่ 1.6(a)) ไปยังคอนฟิกุเรชันที่ต้องการ (รูปที่ 1.6(b)) จะเห็นว่าหุ่นยนต์ทำการเลื่อนและหมุนตัวเพื่อให้สามารถรอดผ่านช่องแคบระหว่างสิ่งกีดขวางไปได้ จากนั้นเมื่อพ้นสิ่งกีดขวางไปแล้ว จึงเลื่อนและหมุนตัวกลับเพื่อไปเรียงตัวตามคอนฟิกุเรชันที่ต้องการ การกำหนดว่าเมื่อใดจะให้หุ่นยนต์เลื่อนตัวหรือหมุนตัวอย่างไรนี้เอง คือแผนการเคลื่อนที่ของหุ่นยนต์ที่เรากล่าวถึง สังเกตได้ว่าเมื่อหุ่นสามารถหมุนตัวได้ คอนฟิกุเรชันของมันจะประกอบด้วย 3 พารามิเตอร์คือค่าพิกัดของจุดอ้างอิงในแนวแกน x ในแนวแกน y และมุม θ ของการเอียงตัวของหุ่น หรือกล่าวอีกนัยหนึ่งเราสามารถนึกถึงคอนฟิกุเรชันว่ามันเป็นจุดในปริภูมิที่มีสามมิติคือ x, y และ θ นั่นก็คือแผนการเคลื่อนที่ของหุ่นก็คือวิถีในปริภูมิสามมิตินี้ที่ลากจากจุดคอนฟิกุเรชันเริ่มต้นไปยังคอนฟิกุเรชันจุดหมาย และดังที่ได้กล่าวไปแล้วว่าบางจุดคอนฟิกุเรชันจะตรงกับการที่หุ่นไปทับกับสิ่งกีดขวาง เราเรียกคอนฟิกุเรชันเหล่านี้ว่าคอนฟิกุเรชันต้องห้าม และเราจะเรียกคอนฟิกุเรชันที่เหลือว่าคอนฟิกุเรชันอิสระ โดยจากนี้ไปเราจะเขียนแทนเซตของคอนฟิกุเรชันต้องห้ามทั้งหมดว่า C-Obstacle (อ่านว่า ซี ออบสตาเคิล) และ แทนเซตของคอนฟิกุเรชันอิสระทั้งหมดว่า C-Free (อ่านว่า ซี ฟรี) ด้วยนิยามใหม่เหล่านี้ เราก็สามารถกล่าวได้อีกนัยหนึ่งว่า การวางแผนการเคลื่อนที่ก็คือการหาวิถีหรือเส้นทางใน C-Free จาก คอนฟิกุเรชันเริ่มต้นไปยังคอนฟิกุเรชันที่ต้องการ



รูปที่ 1.6 แผนการเคลื่อนที่ของสามเหลี่ยมจากคอนฟิกุเรชันใน (a) ไปคอนฟิกุเรชันใน (b)

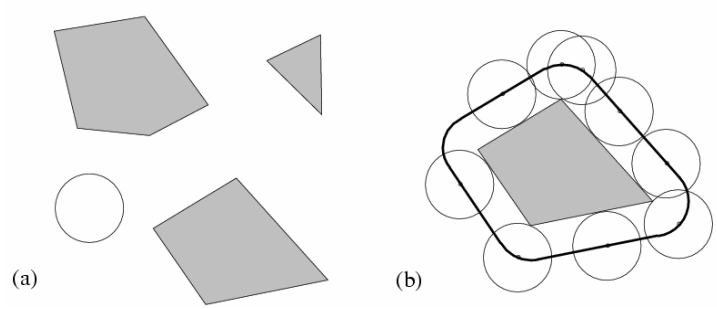

4
3
2
1
(a)
(b)
(c)
(d)
(e)
(f)



รูปที่ 1.7 เส้นทางใน configuration space และการเคลื่อนที่ของวัตถุที่สอดคล้องกันใน workspace

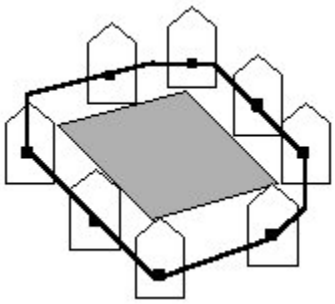
1.4 การวางแผนการเคลื่อนที่ของหุ่นยนต์รูปหลายเหลี่ยม

ก่อนที่จะกล่าวถึงหุ่นยนต์รูปหลายเหลี่ยม ลองพิจารณาหุ่นยนต์รูปแผ่นกลมที่สามารถเคลื่อนที่ได้อย่างอิสระในบริเวณทำงานสองมิติที่มีสิ่งกีดขวางรูปหลายเหลี่ยมดังแสดงในรูปที่ 1.8(a) กำหนดให้จุดศูนย์กลางของแผ่นกลมเป็นจุดอ้างอิง ลองพยายามวาด C-Obstacle โดยการกลิ้งแผ่นกลมไปรอบๆ สิ่งกีดขวางแล้วลากเส้นตามทางเดินของจุดศูนย์กลาง(รูปที่ 1.8(b)) เส้นที่ได้นี้จะล้อมรอบ C-Obstacle ที่เกิดจากสิ่งกีดขวาง เพราะว่าถ้าหากจุดศูนย์กลางของแผ่นกลมล้ำเข้าไปในบริเวณที่ถูกล้อมนี้ ก็แน่นอนว่าแผ่นกลมต้องทับกับสิ่งกีดขวาง หรืออีกนัยหนึ่งก็คือมันได้อยู่ที่คอนฟิกุเรชันต้องห้าม จะเห็นได้ว่ารูปร่างของ C-Obstacle ที่เกิดจากสิ่งกีดขวางแต่ละชิ้น สามารถหาได้จากการขยายขอบของสิ่งกีดขวางชิ้นนั้นๆ ออกไปในแนวตั้งฉากเป็นระยะเท่ากับรัศมีของแผ่นกลม โดยขอบที่ขยายตรงจุดยอดของสิ่งกีดขวางจะเกิดเป็นส่วนวง และดังที่ได้กล่าวไปแล้ว เมื่อเราได้ C-Obstacle ของสิ่งกีดขวางทั้งหมด เราก็สามารถวางแผนการเคลื่อนที่ได้โดยหาเส้นทางของจุดที่ไม่ตัดกับ C-Obstacle ใดๆ และเส้นทางนี้ก็คือเส้นทางของจุดศูนย์กลางของหุ่นแผ่นกลมที่เลื่อนไปได้โดยไม่ชนกับสิ่งกีดขวางใด ความง่ายของการสร้าง C-Obstacle ในกรณีนี้เกิดจากความสมมาตรในทุกทิศทางของรูปแผ่นกลม แต่ถ้าหุ่นยนต์ของเราไม่มีความสมมาตรเหมือนแผ่นกลมล่ะ เราจะคำนวณ C-Obstacle ได้อย่างไร



รูปที่ 1.8 (a) หุ่นยนต์รูปแผ่นกลมในบริเวณทำงานที่มีสิ่งกีดขวางรูปหลายเหลี่ยม และ (b) การสร้าง C-Obstacle จากการกลิ้งแผ่นกลมไปรอบสิ่งกีดขวาง เส้นทึบที่ลากตามจุดศูนย์กลางล้อมรอบ C-Obstacle ที่ต้องการ

กำหนดให้หุ่นยนต์เป็นรูปหลายเหลี่ยมที่หมุนไม่ได้ (เป็นรูปห้าเหลี่ยมในตัวอย่างในรูปที่ 1.9) ลองพิจารณาจุดอ้างอิงจุดหนึ่งบนหุ่นยนต์ในขณะที่หุ่นยนต์เคลื่อนที่เลียบสิ่งกีดขวาง ลากเส้นตามจุดอ้างอิงในขณะที่มันเคลื่อนที่เลียบรอบสิ่งกีดขวางเหมือนในกรณีหุ่นแผ่นกลม เราจะได้เส้นกรอบล้อมรอบบริเวณซึ่งจุดอ้างอิงไม่สามารถล่วงล้ำเข้าไปได้ มิฉะนั้นหุ่นยนต์ต้องทับกับสิ่งกีดขวางนี้ นั่นก็คือบริเวณที่ถูกล้อมรอบอยู่คือ C-Obstacle แต่ว่าเราไม่สามารถสร้าง C-Obstacle ได้โดยการขยายสิ่งกีดขวางเหมือนกรณีหุ่นแผ่นกลม เพราะว่าหุ่นหลายเหลี่ยมนี้ไม่ได้มีความสมมาตรแบบแผ่นกลม ก่อนที่จะกล่าวถึงการคำนวณ C-Obstacle ในกรณีนี้ ลองมาดูกรณีทั่วไปของการสร้าง C-Obstacle โดยใช้ผลรวมมิงคอฟสกี (Minkowski Sum)



รูปที่ 1.9 configuration ต้องห้าม

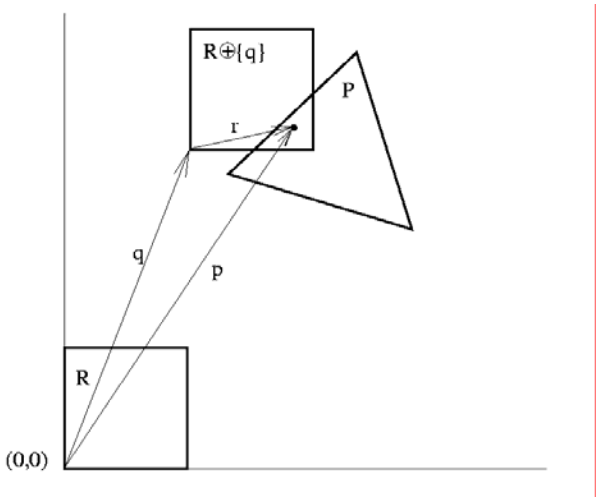
1.4.1 C-Obstacle จากผลรวมมิงคอฟสกี

สำหรับเซตของเวกเตอร์ A และ B ใดๆ นั้น ผลรวมมิงคอฟสกีของ A และ B เขียนแทนได้ด้วย $A \oplus B$ และมีนิยามคือ

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A \ \wedge \ \mathbf{b} \in B\}$$

ซึ่งก็คือเซตของเวกเตอร์ทั้งหมดที่ได้จากการบวกคู่ของเวกเตอร์ใดๆ โดยเวกเตอร์หนึ่งมาจากเซต *A* และอีกเวกเตอร์มาจากเซต *B*

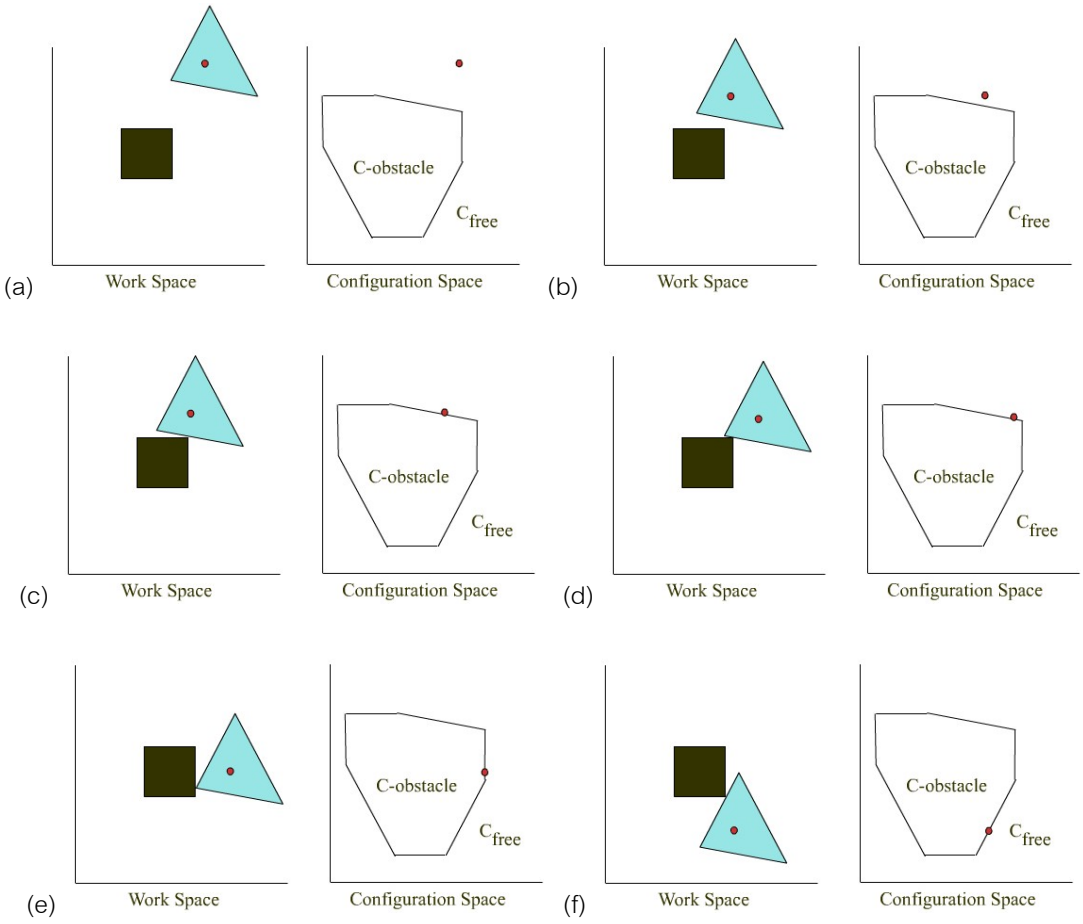
ลองพิจารณาบริเวณทำงานที่มีสิ่งกีดขวางเป็นรูปสามเหลี่ยม กำหนดให้ *P* เป็นเซตของเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดในสิ่งกีดขวางนี้ พิจารณาหุ่นยนต์รูปสี่เหลี่ยมที่เลื่อนได้แต่หมุนไม่ได้ในบริเวณทำงานเดียวกัน ให้จุดอ้างอิงของหุ่นยนต์นี้อยู่ที่มุมซ้ายล่าง โดยให้ *R* เป็นเซตของเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดในตัวหุ่นเมื่อมันอยู่ที่คอนฟิกุเรชัน (0,0) (เมื่อจุดอ้างอิงอยู่ที่จุดกำเนิด) เราจะเห็นได้ว่าเมื่อหุ่นอยู่ที่คอนฟิกุเรชัน q มันจะครอบคลุมจุดที่มีตำแหน่งกำหนดโดยเวกเตอร์ในเซต $\{\mathbf{q}\}\oplus R$ และจากที่ว่าคอนฟิกุเรชันต้องห้ามคือคอนฟิกุเรชันที่หุ่นอยู่ทับกับสิ่งกีดขวาง ทำให้เรารู้ว่า q เป็นคอนฟิกุเรชันต้องห้ามก็ต่อเมื่อมีบางเวกเตอร์ $\mathbf{r} \in R$ และ $\mathbf{p}\in P$ ที่ทำให้ $\mathbf{q}+\mathbf{r}=\mathbf{p}$ หรือเราจะกล่าวได้ว่าสำหรับ คู่เวกเตอร์ $\mathbf{r} \in R$ และ $\mathbf{p}\in P$ ใดๆ จะได้ว่า $\mathbf{q}=\mathbf{p}-\mathbf{r}$ เป็นคอนฟิกุเรชันต้องห้าม นั่นก็คือเราจะได้ว่า C-Obstacle $= P \oplus (-R)$ โดยที่ $-R=\{-\mathbf{a}|\mathbf{a}\in R\}$

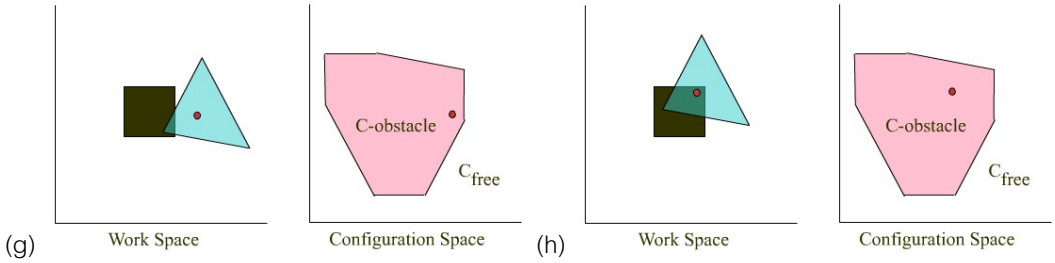


รูปที่ 1.10 forbidden configuration q

นั่นก็คือเราสามารถเปลี่ยนปัญหาการวางแผนการเคลื่อนที่ของหุ่นยนต์รูปหลายเหลี่ยมที่เลื่อนได้แต่หมุนไม่ได้ ในบริเวณทำงานที่มีสิ่งกีดขวางรูปหลายเหลี่ยม ให้เป็นปัญหาการวางแผนการเคลื่อนที่ของหุ่นยนต์จุดในบริเวณทำงานใหม่ โดยที่สิ่งกีดขวาง *P* ในบริเวณทำงานเดิมจะถูกเปลี่ยนให้เป็นสิ่งกีดขวาง $P\oplus(-R)$ ในบริเวณทำงานใหม่โดยที่บริเวณทำงานใหม่ที่กล่าวถึงนี้ ที่จริงแล้วก็คือปริภูมิของคอนฟิกุเรชันของหุ่นยนต์นั่นเอง เราจะเห็นได้ว่าหัวใจสำคัญก็คือการที่เราสามารถเปลี่ยนสิ่งกีดขวางในบริเวณทำงาน ให้เป็น C-Obstacle ในปริภูมิของคอนฟิกุเรชันซึ่งทำให้เราสามารถมองหุ่นยนต์เป็นจุด และทำการวิเคราะห์การเคลื่อนที่ทั้งหมดในปริภูมินี้ ในกรณีที่หุ่นเลื่อนได้แต่หมุนไม่ได้เราจะได้ว่าปริภูมิของคอนฟิกุเรชันมีสองมิติ แต่หากหุ่นยนต์หมุนได้ด้วย ปริภูมิของคอนฟิกุเรชันจะมีสามมิติและมีความซับซ้อนเพิ่มขึ้นมากเนื่องจากรูปร่างของ C-Obstacle เปลี่ยนไปตามแนวการวางตัวของหุ่นยนต์ซึ่งก็เป็นมิติหนึ่ง

ของปริภูมิของคอนฟิกุเรชันที่พิจารณา ขอแย้มว่าเมื่อหุ่นยนต์และสิ่งกีดขวางเป็นรูปหลายเหลี่ยมโดยที่หุ่นยนต์หมุนไม่ได้ เราจะได้ C-Obstacle ที่เป็นรูปหลายเหลี่ยมเช่นกัน ขออุบว่าทำไมจึงเป็นเช่นนี้เอาไว้ก่อน ดังที่ได้กล่าวไปแล้วเมื่อเราได้ C-Obstacle เราก็สามารถนำการวางแผนการเคลื่อนที่ของหุ่นยนต์จุดมาใช้ ในส่วนต่อไปจึงขอกล่าวถึงอัลกอริทึมสำหรับหาเส้นทางสั้นสุดของหุ่นจุดในบริเวณทำงานที่มีสิ่งกีดขวางรูปหลายเหลี่ยม เพื่อแสดงให้เห็นอีกครั้งถึงประโยชน์ของหลักการง่ายๆ ในการวางแผนการเคลื่อนที่ของหุ่นยนต์จุด

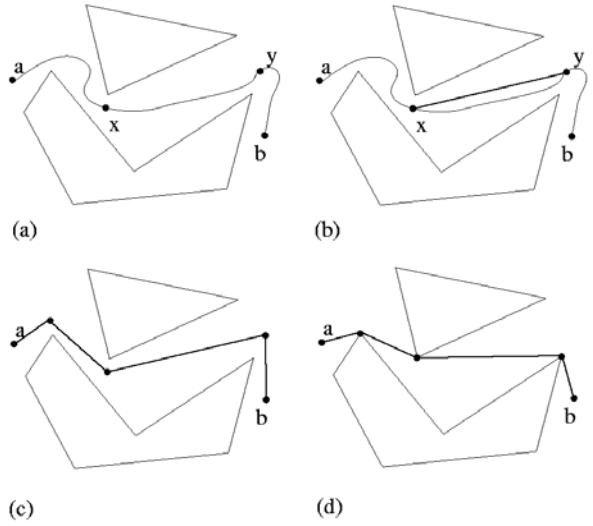




รูปที่ 1.11 configuration space with C-Obstacle

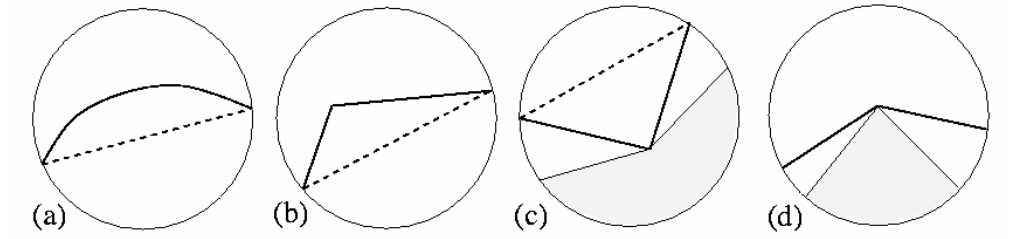
1.5 การหาเส้นทางสั้นสุดด้วยกราฟมองเห็นกัน (visibility graph)

ตอนนี้ขอย้อนกลับไปพิจารณาการวางแผนการเคลื่อนที่ ของหุ่นยนต์จุดในบริเวณทำงานที่มีสิ่งกีดขวางรูปหลายเหลี่ยม จะเห็นได้ว่าอัลกอริทึมที่ใช้แผนผังรูปสี่เหลี่ยมคางหมู ในการวางแผนการเคลื่อนที่นั้น ไม่ได้ให้เส้นทางที่สั้นที่สุดระหว่างจุดเริ่มต้นและจุดหมาย ซึ่งอาจเป็นเส้นทางที่เหมาะสมที่สุดสำหรับบางปัญหา แล้วเส้นทางที่สั้นที่สุดต้องมีลักษณะอย่างไร ลองพิจารณาเส้นทางเดินจากจุด **a** ไปจุด **b** ในรูปที่ 1.12(a) จะเห็นได้ว่าส่วนของเส้นทางที่เป็นเส้นโค้งจากจุด **x** ไปจุด **y** สามารถถูกแทนได้ด้วยเส้นตรงระหว่างสองจุดนี้ (รูปที่ 1.12(b)) ซึ่งก็ชัดเจนว่าสั้นกว่าเส้นทางที่เป็นเส้นโค้ง และนี่ก็เป็นจริงเสมอสำหรับส่วนที่เป็นเส้นโค้งอื่นๆด้วย นั่นคือเส้นทางที่สั้นที่สุดจะต้องประกอบไปด้วยเส้นตรงที่เรียงต่อกันไปจากจุดเริ่มต้นไปยังจุดหมาย คราวนี้พิจารณาเส้นทางในรูปที่ 1.12(c) ที่ประกอบไปด้วยเส้นตรงที่เรียงต่อกันจากจุด **a** ไปจุด **b** เส้นทางนี้ก็ไม่ได้สั้นที่สุด ยังมีเส้นทางที่สั้นกว่าดังแสดงในรูปที่ 1.12(d) ซึ่งเส้นทางนี้เป็นเส้นทางที่สั้นสุดจริงๆ เส้นทางที่สั้นที่สุดจะต้องประกอบไปด้วยเส้นตรงโยงผ่านจุดยอดต่างๆ โดยไม่ทับกับสิ่งกีดขวางใดและจุดยอดเหล่านี้หากไม่เป็นจุดเริ่มต้นหรือจุดปลายก็ต้องเป็นจุดยอดของสิ่งกีดขวาง ทำไมเป็นเช่นนี้ ลองนึกว่าเรามีเชือกที่วางตัวเป็นเส้นโค้งทับกับทางเดินในรูปที่ 1.12(a) หากเราดึงเชือกนี้ให้ตึงโดยที่มีข้อแม้ว่ามันจะต้องผ่านจุด **a** และจุด **b** เราจะได้ว่าเมื่อเชือกตึงสุด (ดึงต่อไม่ได้แล้วเพราะมีสิ่งกีดขวางมากันไว้) เราจะได้ว่าเชือกเส้นนี้จะถูกขึงเป็นเส้นทางในรูปที่ 1.12(d) และเมื่อดึงต่อไม่ได้แล้วก็แสดงว่านี่เป็นเส้นทางที่สั้นสุดที่ผ่านช่องทางนี้



รูปที่ 1.12 เส้นทางจาก a ไป b

เราสามารถอธิบายคุณสมบัติ ที่จำเป็นของเส้นทางสั้นสุดได้อีกวิธี โดยพิจารณาภาพขยายของเส้นทางแบบต่างๆ เพื่อหาเส้นทางที่สั้นกว่าที่สามารถเชื่อมจุดต้นและจุดปลายของเส้นทางที่อยู่ในภาพขยาย จากรูปที่ 1.13(a) เราจะเห็นว่าส่วนของเส้นทางเป็นเส้นโค้งนั้น สามารถแทนด้วยเส้นทางที่เป็นเส้นตรงที่สั้นกว่า ดังที่แสดงในรูปด้วยเส้นประ ส่วนในรูปที่ 1.13(b) แสดงภาพขยายตรงจุดหักมุม ของเส้นทางที่เป็นเส้นตรงเรียงต่อกัน ซึ่งเราก็จะเห็นได้ว่าถ้าตรงจุดหักมุมไม่ใช่จุดยอดของสิ่งกีดขวาง หรือจุดเริ่มต้นหรือจุดปลาย เราก็สามารถเขียนเส้นทางที่สั้นกว่าได้ ในกรณีที่จุดหักมุมเป็นจุดยอดของสิ่งกีดขวาง ก็มีสองกรณีย่อย คือกรณีที่สิ่งกีดขวาง(ส่วนที่แรเงา)ไม่ขวางเส้นทางที่สั้นกว่า (รูปที่ 1.13(c)) และกรณีที่เส้นทางที่สั้นกว่าจะต้องทะลุสิ่งกีดขวางเท่านั้น (รูปที่ 1.13(d))



รูปที่ 1.13 ภาพขยายของเส้นทางแบบต่างๆ

นั่นคือเราจะได้ว่าเส้นทางที่สั้นสุด ต้องเป็นเส้นตรงเรียงกันจากจุดเริ่มต้นไปยังจุดปลาย โดยจะหักมุมได้ที่จุดยอดของสิ่งกีดขวางเท่านั้น และที่ทุกๆ จุดหักมุมนั้น สิ่งกีดขวางจะต้องอยู่ในมุมภายใน เหมือนในรูปที่ 1.13(d) ด้วยคุณสมบัติที่กล่าวมานี้ เราสามารถคำนวณเส้นทางสั้นสุด โดยใช้โครงสร้างที่เรียกว่า ***กราฟมองเห็นกัน*** ซึ่งจุดยอดของกราฟนี้

ประกอบไปด้วยจุดเริ่มต้น จุดปลายและจุดยอดทั้งหมดของสิ่งกีดขวาง โดยที่จะมีเส้นเชื่อมระหว่างจุดยอด **u** และ **v** ในกราฟนี้ก็ต่อเมื่อ **u** และ **v** มองเห็นกัน หรืออีกนัยหนึ่งคือเส้นตรง **uv** ไม่ตัดเนื้อในของสิ่งกีดขวางใดๆ เลย (เส้นขอบของสิ่งกีดขวางก็เป็นเส้นเชื่อมในกราฟด้วยเสมอ) ดังนั้นเมื่อเราได้สร้างกราฟมองเห็นกันสำหรับปัญหาที่พิจารณาแล้ว เราก็สามารถหาเส้นทางสั้นสุด ด้วยอัลกอริทึม Dijkstra สำหรับค้นหาเส้นทางสั้นสุดในกราฟ จากจุดเริ่มต้นไปจุดปลายโดยใช้ระยะทางแบบยุคลิดระหว่างสองจุดยอดของเส้นเชื่อมเป็นระยะทางที่ใช้ในการค้นหา

ถ้ากำหนดให้จำนวนจุดยอดทั้งหมดของสิ่งกีดขวางเป็น n วิธีตรงไปตรงมาในการสร้างกราฟมองเห็นกัน ก็คือพิจารณาแต่ละคู่ของจุดยอด ว่ามองเห็นกันหรือไม่ การตรวจสอบว่ามองเห็นกันหรือไม่ ทำได้โดยดูว่าเส้นเชื่อมจุดยอดคู่นั้นตัดกับเส้นขอบใดของสิ่งกีดขวางหรือไม่ ถ้าตัดก็แสดงว่ามองไม่เห็นกัน เนื่องจากสิ่งกีดขวางมีจำนวนเส้นขอบเป็น n การตรวจสอบสำหรับแต่ละคู่จุดยอดนี้จึงกินเวลา $O(n)$ และการสร้างกราฟ visibility ด้วยวิธีนี้กินเวลาทั้งหมด $O(n^3)$ ... แต่เราทำได้ดีกว่านี้ ด้วยการพิจารณาตรวจสอบการมองเห็นอย่างมีระบบเราจะได้อัลกอริทึมที่ใช้เวลาเพียง $O(n^2 \lg n)$ โครงหลักของของอัลกอริทึมนี้เป็นดังนี้

```
visibilityGraph(S)
Input: เซต S ของสิ่งกีดขวางรูปหลายเหลี่ยม
Output: กราฟ visibility G_V
1:  สร้างกราฟ G=(V,E) โดยที่ V เป็นเซตของจุดยอดของสิ่งกีดขวางในเซต S และ E=∅
2:  สำหรับ ทุกจุดยอด v ∈ V {
3:          W = visibleVertices(v,S)
4:          สำหรับทุกจุดยอด w∈W ให้เพิ่มเส้นเชื่อม (v, w) ลงใน E
5:  }
6:  return G_V
```

โดย visibleVertices ดังที่แสดงในรหัสเทียมที่ตามมานี้ ใช้หลักการที่เรียกว่าการกวาดเส้น (line sweep) โดยการทำงาน จะเหมือนมีเส้นกวาดที่กวาดไปรอบๆ โดยที่ปลายหนึ่งของเส้นกวาดเป็นจุดหมุน ที่อยู่กับที่ตรงจุด **p** และเริ่มต้นกวาดจากแนวขนานแกน $+x$ แล้วกวาดไปรอบๆ ในทิศทางตามเข็มนาฬิกา (เหมือนเรดาร์) ระหว่างที่เส้นกวาดนี้กวาดไปรอบๆ เราก็จะปรับปรุงต้นไม้ค้นได้ดุล T เพื่อให้มันเก็บเส้นขอบที่ตัดกับเส้นกวาดในขณะนั้น ซึ่งก็ชัดเจนว่าเส้นขอบที่ถูกตัดโดยเส้นกวาดมีการเปลี่ยนแปลง ตอนที่เส้นกวาดวางตัวอยู่บนจุดยอดเท่านั้น จึงพอเพียง

ที่เราพิจารณาเฉพาะตำแหน่งที่เส้นกวาดวางตัวบนจุดยอด[1] โดยลำดับการพิจารณาขึ้นกับมุม(วัดตามเข็มนาฬิกา) ระหว่างแกน $+x$ กับรังสีจาก p ไปยังจุดยอด (บรรทัดที่ 1)

```
visibleVertices(p,S)
Input:  เซต S ของสิ่งกีดขวางรูปหลายเหลี่ยมทั้งหมดและจุด p ที่ไม่ได้อยู่ในสิ่งกีดขวางใด
Output: เซต W ของจุดยอดที่ p มองเห็น
1:  เรียงลำดับจุดยอดทั้งหมดของสิ่งกีดขวางใน S ตามมุมที่พิจารณาตามเข็มนาฬิกาจากแกน x ไป
    ยังรังสีจาก p ไปยังแต่ละจุดยอด หากมีมุมเท่ากันจะเรียงให้จุดยอดที่อยู่ใกล้ p มากก่อน และกำ
    หนดให้ w_1, ..., w_n แทนรายการที่เรียงลำดับแล้วนี้
2:  ให้ m เป็นรังสีจาก p ที่ขนานกับแกน +x หาเส้นขอบทั้งหมดของสิ่งกีดขวางที่ตัดกับ m และเก็บ
    เส้นขอบเหล่านี้ไว้ในต้นไม้ค้นได้ดุล( balanced binary search tree) T ตามลำดับที่มันตัดกับ p
3:  W=∅
4:  สำหรับ i = 1 to n {
5:      ถ้า visible( w_i ) จึง เพิ่ม w_i ใน W
6:      เพิ่มเส้นขอบของสิ่งกีดขวางที่วางตัวบน w_i และอยู่ทางด้านตามเข็มนาฬิกาเมื่อเทียบกับ
        รังสี pw_i ลงไปใน T
7:      ลบเส้นขอบของสิ่งกีดขวางที่วางตัวบน w_i และอยู่ทางด้านทวนเข็มนาฬิกาเมื่อเทียบกับ
        รังสี pw_i ออกจาก T }
8:  return W
```

การทดสอบว่า p มองเห็น $\mathbf{w_i}$ หรือเปล่าใน visible ในบรรทัดที่ 5 สามารถทำได้แบบง่ายโดยตรวจสอบว่า $\mathbf{pw_i}$ ตัดกับเส้นขอบที่ตัดกับเส้นกวาดในขณะนั้นหรือเปล่า โดยทำการตรวจสอบกับเส้นขอบที่ใกล้กับ p ที่สุดก็พอ (หาได้จากใบอันซ้ายสุดใน T) แต่หากเป็นไปได้ที่มีสามจุดยอดอยู่บนเส้นตรงเดียวกัน เราสามารถประหยัดได้อีกเล็กน้อยดังในรหัสเทียมต่อไปนี้

```
visible( w_i )
1:  ถ้า i ==1 หรือ w_{i-1} ไม่ได้อยู่บนส่วนเส้นตรง pw_i จึง {
2:      ค้นหาเส้นขอบ e ที่อยู่ที่ใบอันซ้ายสุดใน T
3:      ถ้า เจอ e และ pw_i ตัด e จึง return false แต่ถ้าไม่ return true
4:  } แต่ถ้าไม่
5:      ถ้า p มองไม่เห็น w_{i-1} จึง return false แต่ถ้าไม่ {
```

[1] ตำแหน่งที่มีการเปลี่ยนแปลงที่เราสนใจนี้ เราเรียกว่าจุดเกิดเหตุ (event point)

```
6:                  ค้าหาเส้นขอบ e ใน T ที่ตัดกับ  w_{i-1}w_i
7:                  ถ้า เจอ e จึง return false แต่ถ้าไม่ return true
8:          }
```

1.6 การคำนวณผลรวมมิงคอฟสกี

จากที่กล่าวมาจะเห็นได้ว่า หัวใจสำคัญของการวางแผนการเคลื่อนที่คือการคำนวณ C-Obstacle โดยใช้ผลรวมมิงคอฟสกี ดังนั้นก่อนที่จะจบบทนี้ จะขอทำตามสัญญากล่าวเพิ่มเติมถึงการคำนวณผลรวมมิงคอฟสกี เพื่อใช้หา C-Obstacle ในกรณีที่ทั้งหุ่นยนต์และสิ่งกีดขวางเป็นรูปหลายเหลี่ยม คุณสมบัติที่สำคัญประการแรกที่ขอกล่าวถึงคือ

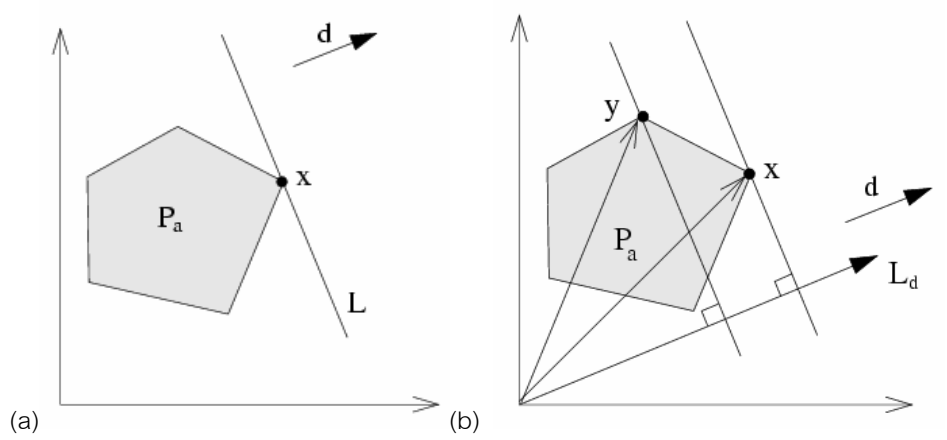
บทตั้งที่ 1.1

กำหนดให้ A และ B เป็นเซตของเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดในรูปหลายเหลี่ยมคอนเวกซ์สองรูป เราจะได้ว่า $A \oplus B$ เป็นเซตของเวกเตอร์ที่ชี้ไปยังจุดทั้งหมดในรูปหลายเหลี่ยมแบบคอนเวกซ์เช่นกัน

พิสูจน์ ให้ $C = A \oplus B$ และสมมุติขัดแย้งว่า C ไม่เป็นเซตคอนเวกซ์ นั่นคือมี $\mathbf{c}_1, \mathbf{c}_2 \in C$ ซึ่งเวกเตอร์ทั้งหมดที่ชี้ไปยังส่วนของเส้นตรง $\mathbf{c}_1\mathbf{c}_2$ ไม่ได้อยู่ใน C ทั้งหมด ให้ $\mathbf{c}_1 = \mathbf{a}_1 + \mathbf{b}_1$ และ $\mathbf{c}_2 = \mathbf{a}_2 + \mathbf{b}_2$ โดยที่ $\mathbf{a}_1, \mathbf{a}_2 \in A$ และ $\mathbf{b}_1, \mathbf{b}_2 \in B$ (เพราะ $\mathbf{c}_1, \mathbf{c}_2 \in C$ จึงต้องมี $\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1, \mathbf{b}_2$ อยู่จริง) กำหนดให้ $\mathbf{c}' = \mathbf{c}_1 + \lambda(\mathbf{c}_2 - \mathbf{c}_1)$ โดยที่ $\lambda \in [0,1]$ เป็นเวกเตอร์ที่ชี้ไปยังจุดบนส่วนของเส้นตรง $\mathbf{c}_1\mathbf{c}_2$ เราเขียนใหม่ได้ว่า $\mathbf{c}' = \mathbf{a}' + \mathbf{b}'$ โดยที่ $\mathbf{a}' = \mathbf{a}_1 + \lambda(\mathbf{a}_2 - \mathbf{a}_1)$ และ $\mathbf{b}' = \mathbf{b}_1 + \lambda(\mathbf{b}_2 - \mathbf{b}_1)$ แต่ว่า $\mathbf{a}' \in A$ และ $\mathbf{b}' \in B$ เพราะ A และ B เป็นเซตคอนเวกซ์ ดังนั้นได้ว่า $\mathbf{c}'$ อยู่ใน $A \oplus B$ □

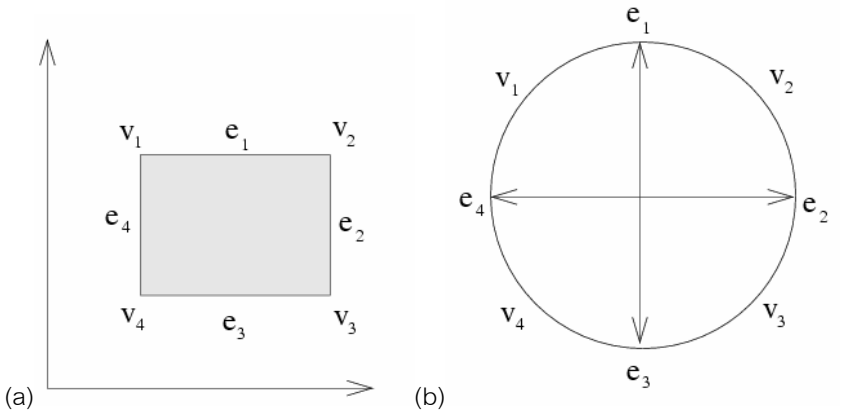
กำหนดให้ A และ B เป็นเซตของเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดในรูปหลายเหลี่ยมคอนเวกซ์ P_A และ P_B ตามลำดับ โดยที่ P_A และ P_B มีเส้นขอบเป็นจำนวน a และ b เส้น วิธีตรงไปตรงมาสุดที่จะหา $A \oplus B$ ก็คือการหาผลบวกทุกคู่ของเวกเตอร์ที่ชี้ไปจุดยอดของ P_A และ P_B แล้วหาเปลือกคอนเวกซ์ (convex hull) ของจุดทั้งหมดที่ชี้โดยเวกเตอร์ผลลัพท์ เซตของเวกเตอร์ทั้งหมดที่ชี้ไปแต่ละจุดในรูปหลายเหลี่ยมคอนเวกซ์ที่ได้ก็คือ $A \oplus B$ ที่ต้องการ แต่ที่จริงเราสามารถคำนวณผลรวมมิงคอฟสกี $A \oplus B$ ได้อย่างมีประสิทธิภาพมากกว่านี้โดยอาศัยการพิจารณาสิ่งที่เราเรียกว่าจุดสุดขีด สำหรับรูปใดๆ จุดสุดขีดของมันในทิศทางตามเวกเตอร์ d ก็คือจุดในรูปนี้ที่อยู่ไกลสุดในทิศทางตามเวกเตอร์ d และเราจะขอกล่าวว่าเวกเตอร์สุดขีดของเซตของเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดในรูปนี้ก็คือเวกเตอร์ที่ชี้ไปยังจุดสุดขีด เมื่อพิจารณาทุกเวกเตอร์ในเซตนี้ ผลคูณภายใน (inner product) ของเวกเตอร์สุดขีดกับ d จะมีค่ามากที่สุด ตัวอย่างในรูปที่ 1.14(a) แสดงจุด x ที่เป็นจุดสุดขีดในทิศทางตามเวกเตอร์ d จะเห็นได้ว่า

หากเราเลื่อนเส้นตรง *L* ซึ่งตั้งฉากกับเวกเตอร์ **d** ผ่านรูปเหลี่ยมในทิศทางตามเวกเตอร์ **d** จุดสุดขีดก็คือจุดสุดท้ายของรูปเหลี่ยมที่เส้นตรง *L* ลากผ่าน และหากเราพิจารณาโปรเจคชันในแนวฉากของเวกเตอร์ที่ชี้ไปยังจุดใน P_a ลงบนเส้น L_d ซึ่งเป็นเส้นจากจุดกำเนิดที่ชี้ไปในทิศทางตามเวกเตอร์ **d** โปรเจคชันของจุด **x** จะมีความยาวมากกว่าโปรเจคชันจากจุดอื่นๆ (เช่นจุด **y** ในรูป) ซึ่งเราก็รู้ว่าความยาวของโปรเจคชันดังกล่าวของจุดหนึ่งๆ ก็คือขนาดของเวกเตอร์ **d** คูณกับผลคูณภายในของเวกเตอร์ที่ชี้ไปที่จุดนี้กับเวกเตอร์ **d** นั่นเอง



รูปที่ 1.14 จุดสุดขีด x ในทิศทาง d

ขอยกตัวอย่างเพื่อประกอบการทำความเข้าใจ พิจารณาจุดสุดขีดของรูปสี่เหลี่ยมผืนผ้าในรูปที่ 1.15(a) ในทิศทางต่างๆ ทั้ง 360 องศา ดังแสดงบนวงกลมในรูปที่ 1.15(b) จุดสุดขีดในทิศทางต่างๆ จะเขียนไว้โดยรอบ เช่นที่ทิศทาง 0 องศา (ขนานแกน +*x*) จุดสุดขีดก็คือจุดบนเส้นขอบ e_2 ทั้งหมด ในทิศทางในช่วงระหว่าง 0 ถึง 90 องศา จุดสุดขีดก็คือจุดยอด v_2 เป็นต้น



รูปที่ 1.15 รูปสี่เหลี่ยมผืนผ้าและจุดสุดขีดในทิศทางต่างๆ

จากนิยามของผลรวมมิงคอฟสกีเราสามารถพิสูจน์ได้โดยไม่ยากว่า

บทตั้งที่ 1.2:

เวกเตอร์สุดขีดในทิศทาง d ของเซต $A \oplus B$ คือผลรวมของเวกเตอร์สุดขีดในทิศทาง d ของเซต *A* และ *B*

พิสูจน์ ให้ $\mathbf{c}$ เป็นเวกเตอร์สุดขีดของ $A \oplus B$ ในทิศทาง $\mathbf{d}$ เราสามารถเขียนได้ว่า $\mathbf{c} = \mathbf{a} + \mathbf{b}$ สำหรับ $\mathbf{a} \in A$ และ $\mathbf{b} \in B$ ความสมมาตรของ $\mathbf{c} = \mathbf{a} + \mathbf{b}$ ทำให้เพียงพอที่จะสมมุติขัดแย้งว่า $\mathbf{a}$ ไม่ใช่เวกเตอร์สุดขีดของ A ในทิศทาง $\mathbf{d}$ นั่นคือมี $\mathbf{a}' \in A$ ซึ่ง $\mathbf{a}' \bullet \mathbf{d} > \mathbf{a} \bullet \mathbf{d}$ เมื่อพิจารณา $\mathbf{c}' = \mathbf{a}' + \mathbf{b}$ ด้วยคุณสมบัติการกระจายของผลคูณภายใน เราได้ว่า $\mathbf{c}' \bullet \mathbf{d} = \mathbf{a}' \bullet \mathbf{d} + \mathbf{b} \bullet \mathbf{d} > \mathbf{c} \bullet \mathbf{d}$ ทำให้ $\mathbf{c}$ ไม่ใช่เวกเตอร์สุดขีดในทิศทาง $\mathbf{d}$ ขัดแย้งกับที่กำหนดไว้ □

และสามารถสรุปได้ว่า

บทตั้งที่ 1.3:

ผลรวมมิงคอฟสกี $A \oplus B$ เป็นเซตของเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดในรูปหลายเหลี่ยมคอนเวกซ์ $P_{A \oplus B}$ ที่มีจำนวนเส้นขอบอย่างมาก *a+b* เส้น

พิสูจน์: การเป็นคอนเวกซ์ของรูปหลายเหลี่ยมของ $A \oplus B$ เป็นไปตามบทตั้งที่ 1.1 สำหรับที่มาของจำนวนเส้นขอบมากสุดนั้น พิจารณาเส้นขอบเส้นหนึ่งของ P_A จะเห็นได้ว่าเวกเตอร์ทั้งหมดจากจุดกำเนิดไปยังแต่ละจุดบนเส้นขอบนี้เป็นเวกเตอร์สุดขีดทั้งหมดในทิศทางตั้งฉากภายนอก (outward normal) ของเส้นขอบเส้นนี้ จากนั้นในทิศทางตั้งฉากนี้หาเวกเตอร์สุดขีดของเซต *B* ก็จะได้เวกเตอร์สุดขีดหนึ่งเวกเตอร์ (หรือไม่ก็เป็นเซตของเวกเตอร์จากจุดกำเนิดไปยังเส้นขอบหนึ่งของ P_B ที่ขนานกับเส้นขอบของ P_A ที่กล่าวถึง) นำเวกเตอร์สุดขีดที่ได้จาก P_A และ P_B ดังกล่าวมาบวกกัน จากบทตั้งที่ 1.2 จะได้เป็นเซตของเวกเตอร์สุดขีดของ $A \oplus B$ ซึ่งเซตผลลัพท์นี้จะกำหนดเส้นขอบหนึ่งเส้นของ $P_{A \oplus B}$ โดยเมื่อเราพิจารณาทุกทิศทางนั่นก็คือพิจารณาทุกเส้นขอบของ P_A เส้นขอบละครั้งก็จะได้ *a* เส้นขอบให้กับ $P_{A \oplus B}$ และเมื่อสลับหน้าที่ของ P_A กับ P_B แล้วพิจารณาทุกเส้นขอบของ P_B ก็จะได้ *b* เส้นขอบให้กับ $P_{A \oplus B}$ แต่ว่าบางเส้นขอบที่ได้นี้อาจซ้ำกับที่หาได้จากการพิจารณาเส้นขอบของ P_A ในกรณีที่มีบางเส้นขอบของ P_A และ P_B ขนานกัน ดังนั้นจำนวนเส้นขอบของ $P_{A \oplus B}$ มีอย่างมากสุด *a+b* □

อัลกอริทึมต่อไปนี้คำนวณผลรวมมิงคอฟสกีของเซตของเวกเตอร์ในรูปหลายเหลี่ยมคอนเวกซ์ P_A และ P_B โดย angle(pq) จะให้ค่ามุมที่เส้นตรง pq ทำกับแกน *x* เมื่อพิจารณาแล้วจะเห็นได้ว่าอัลกอริทึมนี้กินเวลา $O(a+b)$ โดย a และ b เป็นจำนวนจุดยอดของ P_A และ P_B ตามลำดับ

MinkowskiSum(P_A , P_B)

Input: รูปหลายเหลี่ยมคอนเวกซ์ P_A และ P_B โดยมีเวกเตอร์ $\mathbf{v_1}, ..., \mathbf{v}_a$ และ $\mathbf{w_1}, ..., \mathbf{w}_b$ เป็นเวกเตอร์ที่ชี้ไปยังจุดยอดของ P_A และ P_B ตามลำดับโดยเรียงทวนเข็ม กำหนดให้ $\mathbf{v_1}$ และ $\mathbf{w_1}$ เป็นเวกเตอร์ที่ชี้ไปจุดยอดที่มีพิกัดแกน *y* ต่ำสุดในบรรดาจุดยอดทั้งหมดใน P_A และ P_B

Output: รายการเรียงทวนเข็มของเวกเตอร์ที่ชี้ไปจุดยอดของ $P_{A \oplus B}$

1: *i*=1; *j*=1

2: $\mathbf{v}_{a+1} = \mathbf{v_1}$; $\mathbf{w}_{b+1} = \mathbf{w_1}$

3: ทำซ้ำ {

4: เพิ่ม $\mathbf{v}_i + \mathbf{w}_j$ ในรายการจุดยอดของ $P_{A \oplus B}$

5: **ถ้า** angle($\mathbf{v_i v_{i+1}}$) < angle($\mathbf{w_j w_{j+1}}$) **จึง** *i* = *i*+1 แต่ถ้าไม่

6: ถ้า angle($\mathbf{v_i v_{i+1}}$) > angle($\mathbf{w_j w_{j+1}}$) จึง *j* = *j* +1

7: แต่ถ้าไม่ { *i* = *i*+1; *j* = *j* +1; }

8: } **จนกระทั่ง** *i* == *a*+1 **และ** *j* == *b*+1

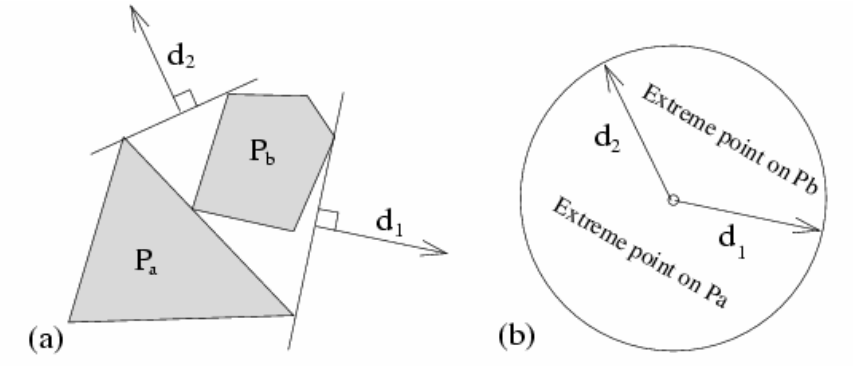
ถึงตรงนี้เราจึงรู้แล้วว่า C-Obstacle ที่เกิดจากหุ่นยนต์รูปหลายเหลี่ยมคอนเวกซ์ *R* และสิ่งกีดขวางรูปหลายเหลี่ยมคอนเวกซ์ *P* มีรูปร่างเป็นรูปหลายเหลี่ยมคอนเวกซ์ซึ่งมีจำนวนเส้นขอบอย่างมาก $n_R + n_P$ โดย n_R และ n_P คือจำนวนเส้นของของหุ่นและสิ่งกีดขวางตามลำดับ แต่โดยทั่วไปเราอาจมีสิ่งกีดขวางหลายชิ้นในบริเวณทำงาน ดังนั้นในการวางแผนการเคลื่อนที่ตามที่ได้อธิบายไปแล้วเราจึงต้องพิจารณายูเนียนของทุกๆ C-Obstacle ที่คำนวณได้จากสิ่งกีดขวางทุกชิ้น แล้วยูเนียนที่ได้จะมีรูปร่างเป็นอย่างไร บอกให้ก่อนว่ายูเนียนที่ได้นี้เป็นรูปหลายเหลี่ยมที่มีจำนวนจุดยอดไม่เกินสองเท่าของผลรวมของจำนวนเส้นขอบของสิ่งกีดขวางทั้งหมด ที่บอกได้ดังกล่าวก็เพราะคุณสมบัติที่ว่า C-Obstacle ทั้งหมดที่ได้นั้นเป็นกลุ่มของแผ่นกลมเทียม (pseudodisc) ต่อไปนี้จะขออธิบายว่าข้อสรุปที่กล่าวถึงนี้มีความเป็นมาอย่างไร

ก่อนอื่นขอยกข้อสังเกตง่ายๆ ต่อไปนี้

บทตั้งที่ 1.4:

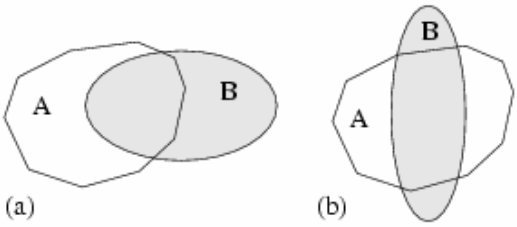
พิจารณาจุดสุดขีดในทุกๆ ทิศทางของยูเนียนของรูปหลายเหลี่ยมคอนเวกซ์สองรูปที่ไม่มีบริเวณภายในทับกัน เราจะได้ว่าหากจุดสุดขีดที่ทิศทาง $\mathbf{d}_1$ และ $\mathbf{d}_2$ อยู่บนรูปหลายเหลี่ยมเดียวกัน จุดสุดขีดก็จะอยู่บนรูปหลายเหลี่ยมนั้นตลอดทั้งช่วงของทิศทาง $[\mathbf{d}_1, \mathbf{d}_2]$ หรือไม่ก็ตลอดช่วงของทิศทาง $[\mathbf{d}_2, \mathbf{d}_1]$

ลองดูตัวอย่างรูปหลายเหลี่ยมคอนเวกซ์ P_a และ P_b ที่ไม่มีบริเวณภายในทับกัน (อาจอยู่ชิดกันได้) ดังในรูปที่ 1.16(a) เราสามารถแสดงว่าจุดสุดขีดจะอยู่บนรูปเหลี่ยมใดตามแผนภาพในรูปที่ 1.16(b) โดยแต่ละจุดบนวงกลมแทนทิศทางจากจุดศูนย์กลางไปยังจุดนั้น จะเห็นได้ว่าเป็นจริงตามบทตั้งที่ 1.4



รูปที่ 1.16 รูปเหลี่ยมคอนเวกซ์สองรูปและจุดสุดขีดในทิศทางต่างๆ

ถึงตรงนี้ขอนิยามแผ่นกลมเทียม รูป *A* และ *B* ในระนาบเป็นแผ่นกลมเสมือนก็ต่อเมื่อ *A-B* และ *B-A* เป็นบริเวณเชื่อมโยง (connected region) เมื่อเป็นบริเวณเชื่อมโยงผลลบจึงเป็นรูปที่มีองค์ประกอบเดียว ดังตัวอย่างในรูปที่ 1.17(b) รูป *A* และ *B* ไม่เป็นแผ่นกลมเทียมเพราะผลลบมีสององค์ประกอบจึงไม่เป็นบริเวณเชื่อมโยง ที่เรียกว่าเป็นแผ่นกลมเทียมก็เพราะลักษณะการตัดกันของเส้นขอบเหมือนกับแผ่นกลม นั่นก็คือสำหรับแผ่นกลมเสมือนสองรูปจะมีการตัดกันของเส้นขอบอย่างมากได้เพียงสองจุดเท่านั้น (เหมือนแผ่นกลมจริงที่เส้นขอบตัดกันเพียงสองจุด ยกเว้นในกรณีที่มีขนาดเท่ากันและทับกันสนิทซึ่งไม่ถือว่าเป็นการตัดกันที่ปกติ) และเราจะขอกล่าวว่ากลุ่มของรูปในระนาบเป็นกลุ่มของแผ่นกลมเทียมเมื่อทุกๆ คู่ของรูปในกลุ่มนี้เป็นแผ่นกลมเทียม



รูปที่ 1.17 รูป A และ B ในระนาบที่เป็น (a) แผ่นกลมเทียม (b) ไม่ใช่แผ่นกลมเทียม

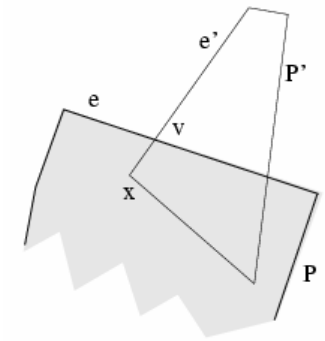
คุณสมบัติที่สำคัญของแผ่นกลมเทียมที่เป็นรูปหลายเหลี่ยมจะใช้ในการวิเคราะห์คือ

บทตั้งที่ 1.5:

กำหนดให้ n เป็นผลรวมของจุดยอดทั้งหมดของกลุ่มของแผ่นกลมเทียมที่เป็นรูปหลายเหลี่ยม จะได้ว่ายูเนียนของแผ่นกลมเทียมทั้งหมดในกลุ่มมีจำนวนจุดยอดไม่เกิน $2n$

<u>พิสูจน์:</u> พิจารณาเส้นขอบของยูเนียนของแผ่นกลมเทียม เราจะเห็นว่าจุดยอดของเส้นขอบของยูเนียนประกอบไปด้วยจุดยอดสองแบบ แบบแรกเป็นจุดยอดของแผ่นกลมเทียมที่มายูเนียนกัน และแบบที่สองเป็นจุดยอดที่เกิดขึ้นใหม่จากการตัดกันของเส้นขอบของแผ่นกลมเทียม เราจะหาจำนวนจุดยอดของเส้นขอบยูเนียนโดยการนับประจุ เริ่มต้นด้วยการให้ประจุหนึ่งประจุกับแต่ละจุดยอดของเส้นขอบของยูเนียนที่เป็นจุดยอดแบบแรก สำหรับจุดยอดแบบที่สอง ดูรูปที่ 1.18 ซึ่งเป็นแผ่นกลมเทียมรูปหลายเหลี่ยม P และ P' จะเห็นได้ว่าจุด **v** เกิดจากการตัดกันของเส้นขอบ *e* กับ *e*' ลองเริ่มที่จุด **v** และไล่ไปตามเส้นขอบ *e* เข้าไปในเนื้อของ *P*' หากปลายของเส้นขอบ *e* หยุดภายใน *P*' เราก็สามารถให้หนึ่งประจุที่แทนจุด **v** กับจุดยอดนี้ได้ แต่ถ้าไม่ (เหมือนกรณีที่แสดงในรูป) พิจารณาที่จุด **v** แล้วลองไล่ตามเส้นขอบ *e*' บ้าง ด้วยคุณสมบัติการเป็นแผ่นกลมเทียมของ *P* และ *P*' ไม่มีทางที่ *e*' จะตัดผ่าน *P* ไปเหมือนที่ *e* ตัดผ่าน *P*' ไป ดังนั้นต้องมีจุดยอดของ *P*' ที่อยู่ภายใน *P* (จุดยอด **x** ในรูป) เราจึงสามารถให้หนึ่งประจุที่แทน **v** กับจุดยอดนี้ หมายความว่าทุกจุดยอดที่อยู่บนขอบของยูเนียน หากเป็นจุดที่เกิดจากการตัดกันของเส้นขอบของแผ่นกลมเทียมแล้ว เราสามารถหาจุดยอดของแผ่นกลมเทียมที่อยู่ภายในยูเนียนเพื่อให้ประจุได้เสมอ และในทางกลับกันหากเราเริ่มต้นที่จุดยอดใดก็ตามของแผ่นกลมเทียมที่อยู่ภายในยูเนียน แล้วไล่ตามเส้นขอบทั้งสองที่ติดกับมันจนกระทั่งพ้นจากเนื้อในของของยูเนียน ก็จะมาถึงจุดยอดซึ่งมีได้สองจุด (จุดตัดของเส้นขอบของแผ่นกลมเทียม) ดังนั้นแต่ละจุดยอดของแผ่นกลมเทียมจะมีได้อย่างมากสองประจุ และด้วยวิธีการให้ประจุที่กล่าวมาจะได้ว่าจำนวนประจุนี้ไม่น้อยกว่าจำนวนจุดยอดบนขอบยูเนียนแน่นอน นั่นก็คือเราจะได้ว่าจุดยอดบนขอบยูเนียนไม่เกิน

$2n$ □



รูปที่ 1.18 จุดยอดจากยูเนียนของแผ่นกลมเทียมรูปหลายเหลี่ยม

ดังนั้นที่เหลือให้พิสูจน์เพื่อให้ได้ข้อสรุปที่กล่าวไปแล้วก็คือ

บทตั้งที่ 1.6:

ให้ P_A และ P_B เป็นรูปหลายเหลี่ยมคอนเวกซ์ที่ไม่มีบริเวณภายในทับกัน ให้ R เป็นรูปหลายเหลี่ยมคอนเวกซ์อีกรูปหนึ่ง เราจะได้ว่าผลรวมมิงคอฟสกี $P_A \oplus R$ และ $P_B \oplus R$ เป็นแผ่นกลมเทียม

<u>พิสูจน์:</u> กำหนดให้ $C_A = P_A \oplus R$ และ $C_B = P_B \oplus R$ เพราะ C_A และ C_B เป็นคอนเวกซ์ แต่ละองค์ประกอบของ $C_A - C_B$ ต้องมีจุดที่อยู่บนเปลือกคอนเวกซ์ของ $C_A \cup C_B$ (แน่นอนว่าจุดนี้ต้องเป็นจุดสุดขีดของ $C_A \cup C_B$ ด้วย) นั่นก็คือหาก $C_A - C_B$ มีมากกว่าหนึ่งองค์ประกอบ เราจะสามารถหาสองทิศทาง $\mathbf{d}_1$ ขององ์ประกอบหนึ่งและ $\mathbf{d}_2$ ของอีกองค์ประกอบ ซึ่งจุดสุดขีดอยู่บน C_A จากบทตั้งที่ 1.2 และนิยามของเวกเตอร์สุดขีด เราได้ว่าเมื่อพิจารณา $P_A \cup P_B$ จะได้ว่าจุดสุดขีดต้องอยู่บน P_A ในทั้งสองทิศทาง $\mathbf{d}_1$ และ $\mathbf{d}_2$ และด้วยบทตั้งที่ 1.4 จุดสุดขีดจะอยู่บน P_A ทั้งช่วง $[\mathbf{d}_1, \mathbf{d}_2]$ หรือไม่ก็ทั้งช่วง $[\mathbf{d}_2, \mathbf{d}_1]$ ด้วยการใช้บทตั้งที่ 1.2 และนิยามของเวกเตอร์สุดขีดอีกครั้ง เราจะได้ว่าจุดสุดขีดอยู่บน C_A ในช่วงของทิศทางที่ต่อเนื่องช่วงเดียว ซึ่งจะเห็นได้ว่าเป็นไปไม่ได้ที่ $C_A - C_B$ มีมากกว่าหนึ่งองค์ประกอบ □

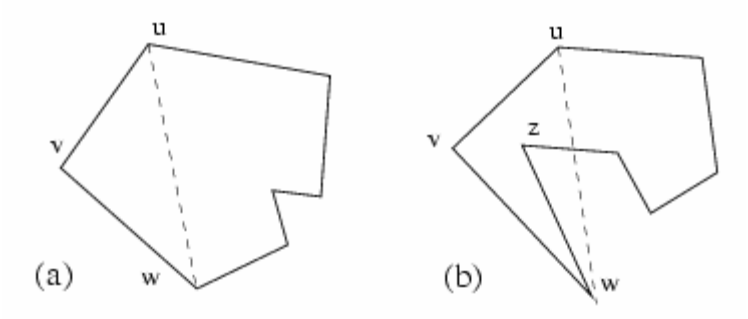
ที่ผ่านมาเรากล่าวถึงการคำนวณผลรวมมิงคอฟสกีของเซตคอนเวกซ์เท่านั้น แล้วหากสิ่งกีดขวางหรือหุ่นยนต์ไม่เป็นคอนเวกซ์ล่ะ ปัญหานี้ตอบไม่ยากด้วยคุณสมบัติการกระจายบนยูเนียนของผลรวมมิงคอฟสกีที่ว่า

$$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$$

นั่นก็คือหากหุ่นยนต์เป็นรูปหลายเหลี่ยมคอนเวกซ์แต่สิ่งกีดขวางไม่เป็นรูปคอนเวกซ์เราสามารถหั่นสิ่งกีดขวางออกเป็นรูปคอนเวกซ์ย่อยๆ แล้วใช้วิธีคำนวณผลรวมมิงคอฟสกีที่กล่าวถึงไปแล้วจากนั้นจึงนำมายูเนียนกัน วิธีหั่นเป็นรูปคอนเวกซ์ย่อยๆ อย่างง่ายวิธีหนึ่งก็คือการหั่นเป็นสามเหลี่ยม ซึ่งสำหรับรูปหลายเหลี่ยมที่มีเส้นขอบ *n* เส้นเราจะสามารถหั่นได้เป็นสามเหลี่ยม *n-2* รูป ซึ่งถ้าเราหาผลรวมมิงคอฟสกีของรูปหลายเหลี่ยมนี้กับรูปเหลี่ยมคอนเวกซ์ที่มีเส้นขอบ *m* เส้น ผลรวมมิงคอฟสกีที่ได้จะมีความซับซ้อนเป็น $O(nm)$ ลองคิดดูว่าทำไมและคิดต่อไปด้วยว่าผลรวมมิงคอฟสกีของเซตที่ไม่เป็นคอนเวกซ์ทั้งคู่จะมีความซับซ้อนเท่าไร ก่อนจะจบเรื่องของผลรวมมิงคอฟสกีขอพิสูจน์ให้เห็นว่ารูปหลายเหลี่ยมที่มีเส้นขอบ *n* เส้นสามารถแบ่งเป็นรูปสามเหลี่ยม *n-2* รูปเสมอ

<u>พิสูจน์:</u> ใช้อุปนัยเชิงคณิตศาสตร์ เริ่มด้วยกรณีฐานเมื่อรูปเหลี่ยมมีสามด้านเราได้ว่ามันก็คือรูปสามเหลี่ยมหนึ่งรูปตรงตามสมมุติฐานที่พิสูจน์ เราสมมุติต่อว่าสมมุติฐานนี้เป็นจริงสำหรับรูปเหลี่ยมที่มีตั้งแต่ 3 ด้านไปถึง *n*-1 ด้าน ดังนั้นก็เพียงพอที่เราจะแสดงให้ดูว่าเราสามารถแบ่งรูปเหลี่ยมที่มี *n* ด้านเป็นรูปหลายเหลี่ยมสองรูปที่มีจำนวนด้านน้อยกว่า *n* สำหรับรูปหลายเหลี่ยมรูปหนึ่ง พิจารณาจุดยอดที่อยู่ซ้ายสุด (หากมีหลายจุดให้เลือกจุดต่ำสุด) เรียกมันว่าจุดยอด v ทั้งสองเส้นขอบที่ติดกับมันจะมีจุดยอดปิดปลาย เรียกจุดยอดทั้งสองว่า u และ w พิจารณาสามเหลี่ยม uvw หากในสามเหลี่ยมนี้ไม่มีจุดยอดใดอยู่ภายใน เราสามารถแบ่งรูปหลายเหลี่ยมที่ให้มาออกเป็นรูปหลายเหลี่ยมที่เล็กลงโดยแบ่งตามเส้น uw (ดูรูปที่ 1.19) แต่ว่าหากภายในสามเหลี่ยม uvw มีจุดยอดอยู่ภายใน เราสามารถเลือกจุดยอดที่อยู่ห่างเส้น uw มากที่สุดได้และขอเรียกมันว่าจุดยอด z ด้วยการเลือกแบบนี้เราแน่ใจได้ว่าเส้น vz จะไม่ตัด

กับเส้นขอบใดเลย ดังนั้นเราสามารถแบ่งรูปหลายเหลี่ยมที่กำหนดให้ออกเป็นรูปหลายเหลี่ยมสองรูปด้วยการแบ่งตามเส้น vz จะเห็นได้ว่ารูปหลายเหลี่ยมสองรูปเล็กมีจำนวนด้านเป็น $n - k + 1$ และ $k + 1$ ซึ่งตามสมมุติฐานจะมีจำนวนสามเหลี่ยมทั้งหมด $[(n - k + 1) - 2] + [(k + 1) - 2] = n - 2$ □



รูปที่ 1.19 การแบ่งรูปหลายเหลี่ยมเป็นสองรูปที่เล็กลง