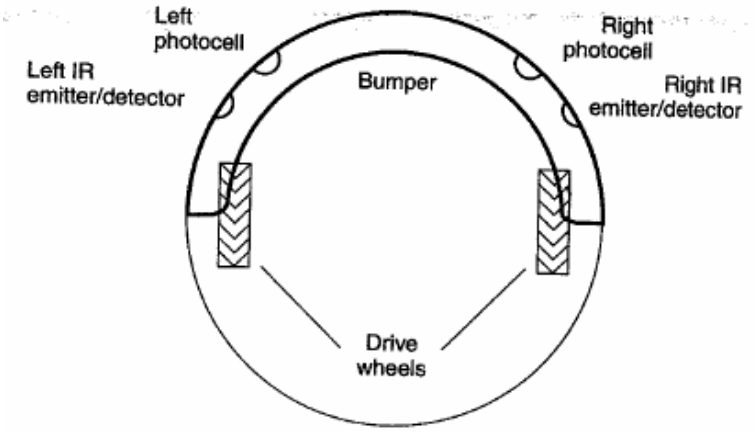
รู้จักกับหุ่นยนต์

อยากให้ทุกคนได้เริ่มทำความรู้จักกับหุ่นยนต์อย่างรวดเร็ว วิธีที่ดีที่สุดคือได้เล่นกับหุ่นยนต์จริงๆ แต่น่าเสียดายที่หลายคนมีอาการแขยงอะไรที่เป็น hardware วิธีที่ดีน้อยกว่าจึงดูเหมือนจะเหมาะกว่า สะดวกกว่า เพราะน่าจะเข้าถึงได้ง่ายกว่ามาก พวกเราโชคดีมาก (หรือเปล่า?) ที่อยู่ในยุคที่มีอุปกรณ์ช่วยลดแรงสมองที่ต้องใช้ในการจินตนาการ เรามีคอมพิวเตอร์ เราสามารถจำลอง นี่ นั่น โน่น แล้วให้ได้เห็นกับตาว่ามันทำงานอย่างไร ทุกคนคงคุ้นเคยกันดีกับเรื่องนี้เพราะได้ใช้ simulator มาแล้วอย่างโชกโชน เช่นในวิชา digital logic จึงไม่จำเป็นที่ต้องอธิบายเพิ่มว่า simulator คืออะไร แต่ถึงแม้ simulator จะเป็นเครื่องมือชิ้นสำคัญ มันก็เป็นเพียงโปรแกรมที่เขียนขึ้นเพื่ออธิบาย หรือเพื่อขยายผลจากจินตนาการของใครบางคน มันไม่ใช่ของจริง! อาจจะดูเหมือนขัดแย้งที่วิชา robotics ซึ่งเป็นเรื่องเกี่ยวข้องกับของจริงๆ ต้องพึ่งพาสิ่งที่เป็นของปลอมอย่าง simulator แต่ก็เป็นเรื่องที่หลีกเลี่ยงไม่ได้ที่จะต้องใช้มัน คำถามยอดนิยมของงานที่มีการใช้ simulator ก็คือว่า simulator นี้สมจริงหรือไม่อย่างไร ประเด็นก็คืออย่าเชื่อ simulator อย่างงมงาย Ron Arkin นักหุ่นยนต์ชื่อดังยังยอมรับใน [ref1]] ว่า “Simulator, unfortunately, is a necessary evil.”

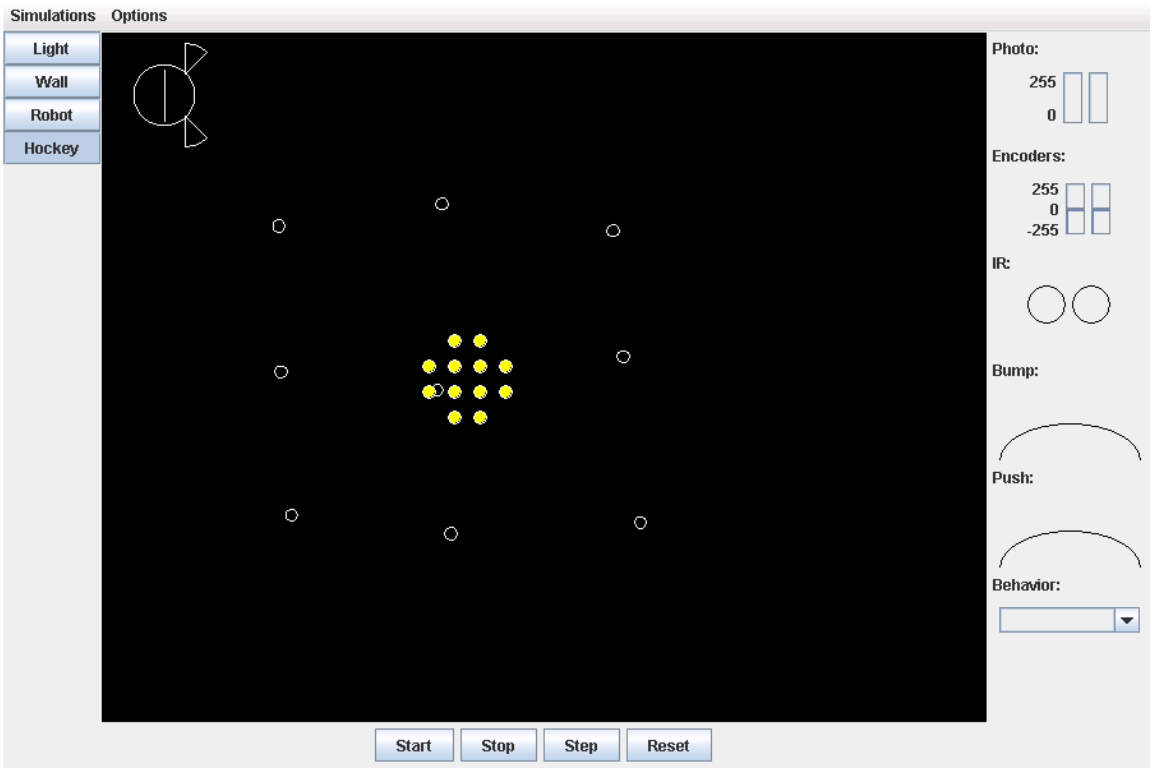
จะขอเริ่มสาธิตหุ่นยนต์ใน simulator หุ่นยนต์ที่ใช้ (มองจากด้านบน) จะเป็นดังรูป



หุ่นยนต์ตัวนี้มีสองล้อขับเคลื่อน (drive wheel) แต่ละล้อมีมอเตอร์คุมการหมุนแยกกัน ด้านละหนึ่งตัว (ของจริงถ้ามีเพียง 2 ล้อ หุ่นยนต์จะกระดกได้ จึงมีอีกหนึ่งหรือสองล้อที่หมุนได้อิสระ ไม่มีมอเตอร์ติด ล้อแบบที่ว่านี้เรียกว่า caster) เราเรียกระบบขับเคลื่อนแบบนี้ว่า differential drive เรื่องของล้อและการควบคุมล้อยังมีให้บรรยายได้อีกยาว ขอเก็บไว้ก่อน ที่ด้านหัวของหุ่นยนต์มีอุปกรณ์ที่เรียกว่า bumper อุปกรณ์ตัวนี้จะบอกหุ่นยนต์

ว่ามันชนอะไรหรือเปล่า การทำงานของ bumper เหมือนสวิท (switch) แบบปกติเปิดที่เราคุ้นเคย นั่นคือเมื่อมีอะไรมากด วงจรถึงจะปิด bumper เป็นอุปกรณ์หุ่นยนต์ชิ้นสำคัญระดับเป็นตาย หุ่นยนต์ที่ไม่รู้ว่ามันชนอะไรอยู่อาจพยายามฝืนบังคับให้ล้อหมุนด้วยความเร็วที่ต้องการ หากมันกำลังชนกำแพง มันก็จะจ่ายไฟเพิ่มให้ล้อเรื่อยๆ เพราะคิดว่ายังจ่ายไฟไม่พอให้ล้อหมุนด้วยความเร็วที่ต้องการ จนอาจทำให้ไหม้เสียหายได้หากไม่มีการออกแบบป้องกันไว้ เรื่องพวกนี้ simulator ส่วนใหญ่ไม่มี เรียนรู้ได้จากของจริง การออกแบบ bumper ที่ดีไม่ใช่เรื่องง่ายเลย ลองคิดว่าถ้าคุณต้องประดิษฐ์ bumper ของจริงเพื่อพันรอบตัวหุ่นยนต์ ให้ตรวจสอบได้ว่าหุ่นยนต์ชนอะไรหรือเปล่า จากมุมใหนก็ได้ เราจะทำอย่างไรดี อุปกรณ์อีกอย่างคือ photo cell จะติดไว้สองตัวทางด้านหน้าซ้าย และหน้าขวา เป็นอุปกรณ์ sensor (ตัวตรวจรู้) สำหรับวัดความเข้มของแสง คือถ้าบริเวณที่หุ่นยนต์อยู่สว่างมากก็จะได้ค่ามาก สว่างน้อยก็ได้ค่าน้อย ในทางปฏิบัติเราอาจใช้ photo transistor หรือ LDR (light dependent resistor) ก็ได้ และแน่นอนว่าอุปกรณ์อีเลคโทรนิคเหล่านี้จะให้ค่าที่วัดได้เป็นสัญญาณไฟฟ้า เราต้องใช้ A/D converter เพื่อเปลี่ยนสัญญาณไฟฟ้าให้เป็นค่าดิจิทัลสำหรับไปประมวลผลด้วยคอมพิวเตอร์ อุปกรณ์สุดท้ายบนหุ่นยนต์คือ IR detector มีอยู่สองชุด ที่ด้านหน้าซ้าย และด้านหน้าขวา ทำหน้าที่ตรวจสอบว่ามีอะไรขวางลำแสง infrared (IR) อยู่หรือเปล่า ถ้ามี IR จะถูกสะท้อนกลับและตรวจจับได้ อุปกรณ์ชิ้นนี้จัดเป็นอุปกรณ์สำหรับหลีกเลี่ยงการชน (collision avoidance) เพราะสามารถตรวจว่ามีอะไรขวางอยู่ ยังไม่ทันได้ชน แต่ bumper นั้นจัดเป็นอุปกรณ์สำหรับตรวจจับการชน (collision detection) มันจะบอกได้ว่ามีการชนจริงๆ เกิดขึ้นหรือไม่ จะเห็นได้ว่าเรื่องชนเป็นเรื่องใหญ่ ได้รับความสนใจมาก นี่ก็เพราะการชนเป็นสิ่งที่ไม่พึงปรารถนาเลยสำหรับนักหุ่นยนต์ หุ่นยนต์ไม่ชอบการกระแทก เพราะมันอาจทำให้เกิดความเสียหายกับทั้งกลไกและวงจรได้ ถ้าใครเคยได้ดูเทปการทดลองหุ่นยนต์จะเห็นได้ว่า บ่อยครั้งจะมีนักเรียนที่ทำหน้าที่เป็น safety team จดๆ จ้องๆ ถ้าหุ่นยนต์ล้มก็พร้อมจะพุ่งรับ (ถ้าหุ่นยนต์พัง อาจมีนักเรียนปริญญาเอกเรียนไม่จบ) การออกแบบหุ่นยนต์ให้สามารถทำงานได้ในโลกจริงต้องเตรียมรับมือกับสถานการณ์ต่างๆ รวมทั้งการกระแทก การชน นี่เป็นเรื่องที่ท้าทายสำหรับนักหุ่นยนต์ และโชคดีที่เรายังไม่ต้องเป็นห่วงในเรื่องเหล่านี้ในการทดลองด้วย simulator ของเรา

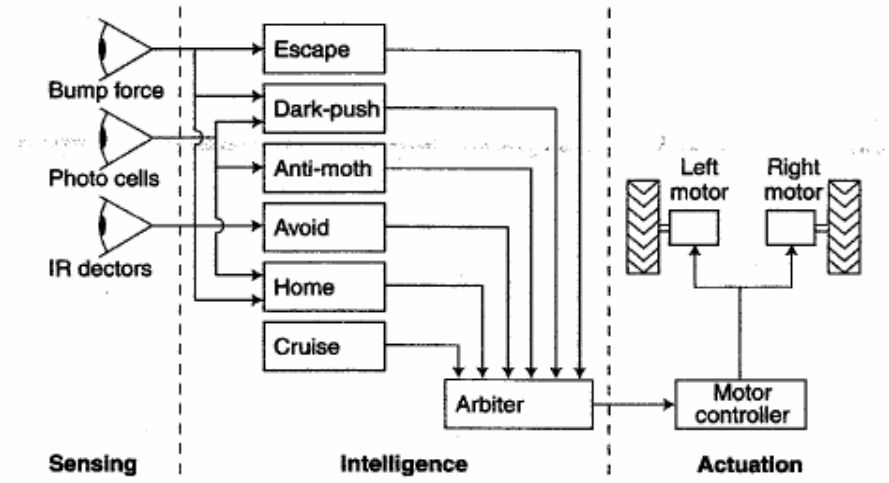
ขอให้ทุกคนเปิด bsim.html เพื่อรัน BSim applet (ต้องมี java runtime environment บนเครื่อง) จากนั้นให้คลิกเลือก [Simulation, Collection] แล้วคลิก [Start] จะเห็นหน้าจอเป็นดังรูป



แถบ display ทางขวาแสดงค่าจาก sensor โดย ค่า Photo แสดงค่าความเข้มแสงที่ได้จาก photo cell ค่า IR แสดงว่า IR sensor ตรวจว่ามีอะไรขวางอยู่หรือไม่ ส่วนค่า Bump แสดงว่า bumper ตรวจจับว่ามีการชนเกิดขึ้นหรือไม่ ที่บรรยายไปทั้งหมดเป็น sensor ที่ตรวจสอบสิ่งแวดล้อมภายนอก(หุ่นยนต์) แต่ sensor อีกตัวคือ encoder จะให้ค่าที่อ่านได้จากตัวหุ่นยนต์ ค่า Encoders แสดงความเร็วของล้อทั้งสอง ความเร็วเป็นบวกคือหมุนไปข้างหน้า เป็นลบคือหมุนไปข้างหลัง ลองสังเกตค่า encoders ขณะหุ่นยนต์เคลื่อนที่ไปข้างหน้า ข้างหลัง และขณะหมุนทั้งทวนเข็มและตามเข็ม ต้องขอแทรกตรงนี้ว่า การอธิบายการเคลื่อนที่ของหุ่นยนต์โดยพิจารณาความเร็วของล้อเป็นเรื่องที่ต้องระวัง สำหรับหุ่นยนต์สองล้อของเรา ในโลกจริง แม้ว่าสมมุติให้สองล้อหมุนไปข้างหน้าด้วยความเร็วเท่ากันพอดี (ซึ่งก็ทำไม่ง่าย) หุ่นยนต์ก็อาจไม่เคลื่อนที่ไปเป็นเส้นตรงตลอดก็ได้ เพราะหากขนาดของล้อต่างกันเพียงเล็กน้อย หุ่นยนต์ก็จะวิ่งเป็นเส้นโค้งแล้ว (ล้อที่ทำมาเท่ากันจากโรงงาน เมื่อผ่านการใช้งาน ก็อาจเกิดการสึกหรอได้) นอกจากนี้หุ่นยนต์ก็อาจวิ่งบนพื้นผิวที่หลากหลาย บางครั้งล้อ(อาจเพียงข้างเดียวก็ได้)อาจเกิดการหมุนเปล่า เช่นขณะวิ่งบนพื้นลื่นมาก การทำนายการเคลื่อนที่ของหุ่นยนต์ในโลกจริงให้แม่นโดยพิจารณาเพียงความเร็วของล้อจึงเป็นไปไม่ได้ในทางปฏิบัติ และนี่ก็เป็นอีกหนึ่งของความท้าทายของงานหุ่นยนต์ และสิ่งที่ต้องระลึกไว้ขณะใช้ simulator

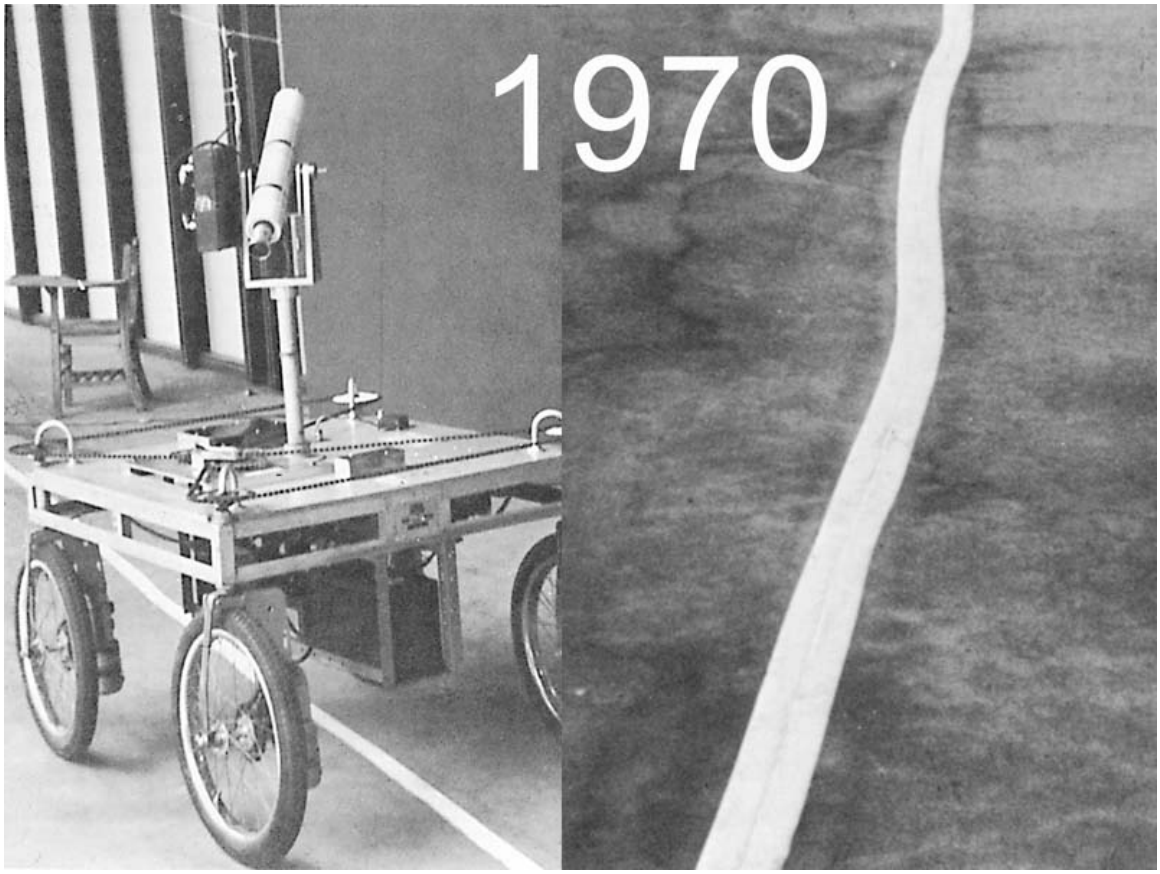
เราจะเห็นหุ่นยนต์เคลื่อนที่ไปมา โดยมีกระจุกของจุดเหลืองอยู่กลางบริเวณทำงาน จุดเหลืองพวกนี้คือหลอดไฟ เมื่อหุ่นยนต์เข้าใกล้หลอดไฟ photo cell จะได้ค่าความสว่างของแสงเพิ่มขึ้น และที่เราเห็นเป็นวงกลมเล็กๆ (ไม่ระบายสี) กระจายอยู่ทั่ว คือลูกฮอคกี้ (ภาษาอังกฤษเรียกว่า puck) หน้าที่ของหุ่นยนต์ตัวนี้ก็คือการเก็บลูกฮอคกี้ที่กระจายอยู่เกลื่อน มารวมไว้ตรงกลาง ลองดูหุ่นยนต์ทำงานสักห้านาที ให้สังเกตุว่าหุ่นยนต์ทำงานอย่างไร มีพฤติกรรมอย่างไร เช่นเวลาชนกำแพง มันทำอย่างไร เวลาชนลูกฮอคกี้ มันทำอย่างไร มันทำงานของมันได้หรือไม่?

จะเห็นได้ว่าหุ่นยนต์ของเราสามารถทำงานได้ตามเป้าหมาย ถึงแม้ว่างานจะคืบหน้าไปอย่างช้าๆ บางครั้งเก็บลูกฮอคกี้ได้ บางครั้งหลุด ทำสำเร็จบ้าง พลาดบ้าง แต่โดยรวมก็ทำงานเสร็จจนได้ ระบบที่ออกแบบยอมให้ทำผิดได้บ้างมีความทนกว่าระบบที่ต้องทำถูกเสมอ ระบบที่ต้องทำถูกต้องเสมอต้องเตรียมคำตอบสำหรับทุกสถานการณ์ บางครั้งระบบที่เตรียมไว้ก็เกิดความขัดแย้งกันเองเมื่อต้องทำงานในสถานการณ์ที่ไม่คาดคิด ยิ่งมีการเตรียมมากวิธี ความซับซ้อนของระบบก็ยิ่งมากเป็นทวีคูณ ทำให้โอกาสผิดพลาดมาก หุ่นยนต์เก็บฮอคกี้ของเราทำงานแบบที่ขอเรียกว่าไม่คิดมาก ฝรั่งเรียกว่า behavior based architecture โดยสามารถเขียนเป็นแผนผังต่อไปนี้

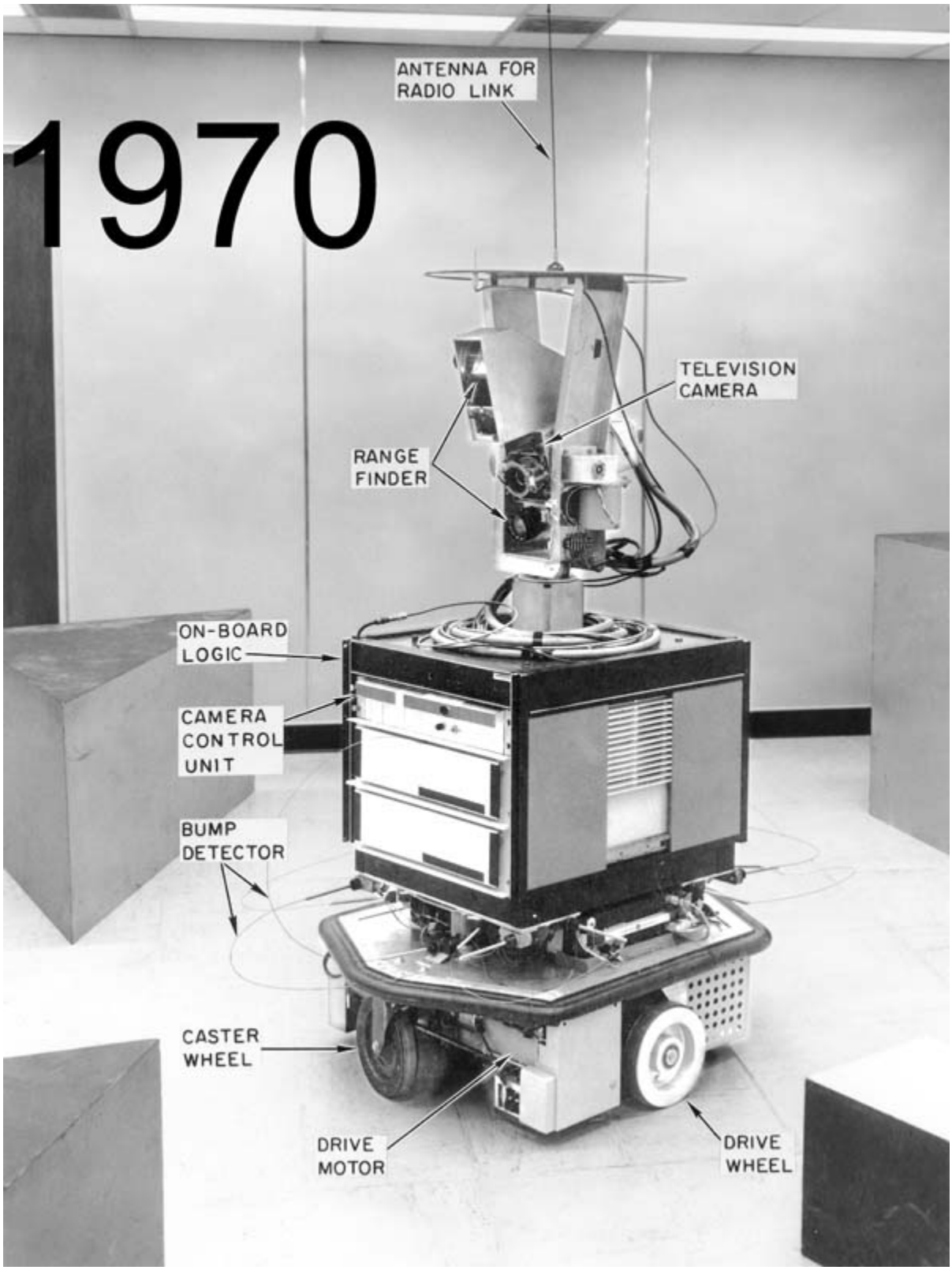


ระบบนี้ก็ดูมีส่วนประกอบเหมือนหุ่นยนต์ทั่วไป นั่นคือประกอบไปด้วยสามส่วนหลักคือ ส่วนตรวจรู้ (sensing) ส่วนคิด (intelligence) และส่วนทำงาน (actuation) จุดเด่นของระบบแบบไม่คิดมากก็คือต้องการผลักดันให้ผลของการตรวจรู้ กลายไปเป็นการทำงานให้เร็วที่สุด ซึ่งนั่นก็คือทำให้ส่วนคิดสั้นสุด ในแผนภาพนี้จะเห็นว่าส่วนคิดจะประกอบด้วย module ที่รับข้อมูลโดยตรงจาก sensor อยู่ห้าตัว แต่ละตัวทำงานเป็นอิสระกัน ลองดูในreadme ของ BSim จะรู้ว่าแต่ละ module ทำงานง่ายๆ ทั้งนั้น แต่ด้วยความที่มันทำงานเป็นอิสระต่อกัน ในบางโอกาส มากกว่าหนึ่ง module อาจต้องการสั่งงานล้อพร้อมกัน หรือเกิดความขัดแย้งในการขอใช้ทรัพยากร เรา

จึงต้องมี module พิเศษทำหน้าที่เป็นแม่บ้านคอยจัดการว่าใครจะได้ใช้ล้อ module ที่ว่านี้ก็คือ arbiter วิธีง่ายๆ ของ arbiter ก็คือให้มีการจัดเรียงระดับความสำคัญของ module ต่างๆ เมื่อมีการแย่งกันใช้ล้อ ก็ให้ตัวที่มีความสำคัญกว่าใช้ก่อน ฟังดูเหมือนง่ายๆ ไม่มีอะไร แต่แท้จริงมีอะไรลึกซึ้งแฝงอยู่ ต้องย้อนไปพิจารณาแนวคิดในการสร้างหุ่นยนต์ในยุคแรกๆ เจ้าหุ่นยนต์สองตัวในรูปข้างล่างคือ Shakey และ Stanford Cart เป็นตัวอย่างจากยุคปี ค.ศ.1960-1970 หุ่นสองตัวนี้ทำงานด้วยการใช้เซนเซอร์เก็บข้อมูลสิ่งแวดล้อมให้มากที่สุดเท่าที่เป็นไปได้ จากนั้นมันจะส่งข้อมูลเหล่านี้ด้วยสัญญาณคลื่นวิทยุให้คอมพิวเตอร์เมนเฟรมทำการวิเคราะห์สร้าง model ของโลกรอบๆ ตัวมัน แล้วทำการวางแผนว่าจะทำอะไรต่อไปดีจึงตรงกับหน้าที่ที่ได้รับมอบหมาย ทุกครั้งที่มีการเคลื่อนที่นิดนึงก็ต้องทำกระบวนการนี้ซ้ำใหม่ อย่างตัว Stanford Cart นี่ทดลองเคลื่อนที่หลบสิ่งกีดขวางในห้องเป็นระยะ 30 เมตรใช้เวลาไป 5 ชั่วโมง การสร้าง world model นี้เป็นเรื่องใช้เวลามาก และไม่สามารถทำได้แม่นยำ เพราะความไม่ครบ และไม่แม่นของเซนเซอร์ และด้วยความเร็วของคอมพิวเตอร์สมัยนั้น แต่ละขั้นตอนจึงใช้เวลามากจริงๆ ตรงนี้แหละที่ behavior based robotics ถือเป็นโอกาส หัวใจของวิธีนี้ก็คือการไม่ต้องสร้าง world model แต่ใช้ข้อมูลจาก sensor มาเปลี่ยนเป็น action ให้เร็วที่สุด วิธีนี้เชื่อว่าโลกเป็นตัวแทนที่ดีที่สุดของโลกเอง แทนที่ต้องเปลี่ยนโลกให้เป็น symbol ในคอมพิวเตอร์ก่อน ทำไมไม่ใช้ตัวโลกเองไปซะเลยล่ะ !



Stanford Cart was an early (1970s) example a somewhat autonomous vehicle. It could be remotely operated, but would also follow a white line. A prototype vision system added in 1979 enabled it to cross a thirty-meter room dotted with obstacles. Travel time: a whopping 5 hours.

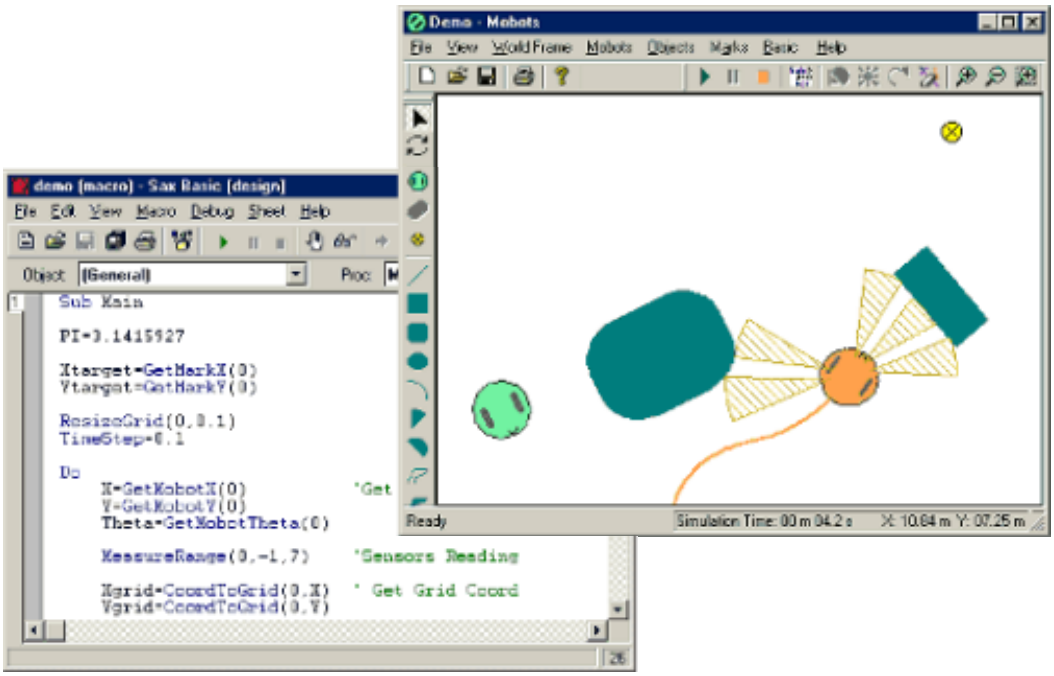


Shakey

วิดีโอ Shakey

วิดีโอ Stanford cart

การทำงานแบบไม่คิดมากที่แสดงให้เห็นใน bsim อาจมีประสิทธิภาพที่ดี แต่ก็จัดว่าสามารถทำงานได้ ลองดูอีกตัวอย่างที่ใช้หลักการไม่คิดมากเหมือนกัน แต่เป็นการควบคุมหุ่นให้วิ่งไปจุดหมาย พร้อมกับหลบสิ่งกีดขวาง ลองเปิดโปรแกรม mobotsim แล้วเปิดไฟล์ VFF แล้วลอง run และสังเกตุ ลองเปิดดู source code ภาษา basic ที่ใช้ควบคุมหุ่นยนต์ด้วย ลองเล่นดูแล้วหาว่าจุดอ่อนของวิธีนี้อยู่ที่ไหน



mobotsim 1.0

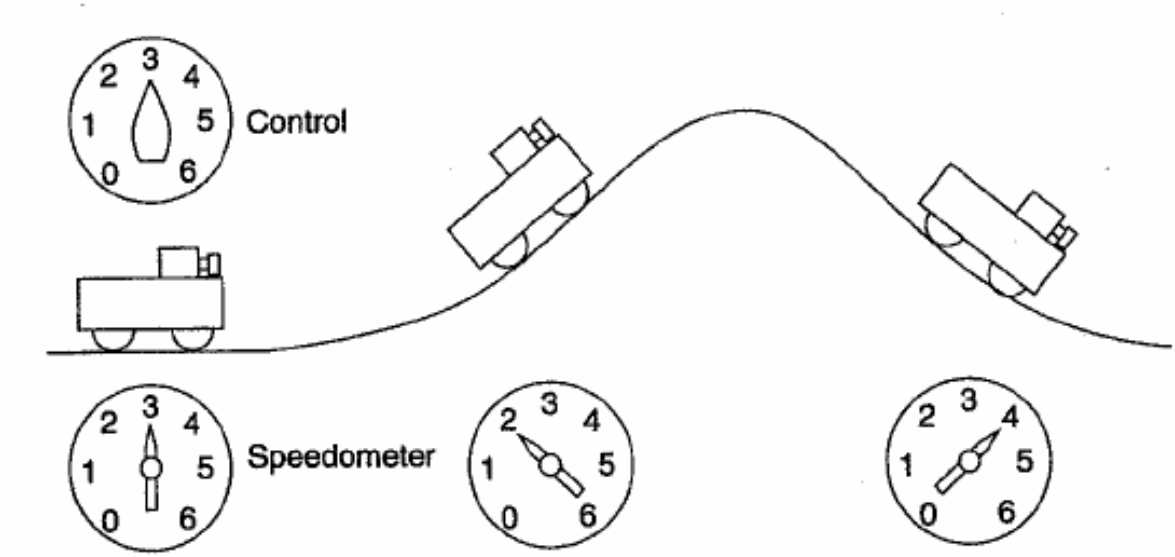
หุ่นยนต์ตัวนี้มีความสามารถพิเศษคือการรู้ตำแหน่งตัวเองในโลกจริง (อุปกรณ์เช่น GPS สามารถให้ข้อมูลตำแหน่งได้) เมื่อเรากำหนดตำแหน่งหรือพิกัดของเป้าหมาย หุ่นยนต์จะใช้หลักการที่เรียกว่าแรงเทียม (artificial force field) ในการควบคุมการเคลื่อนที่ของหุ่น เป้าหมายจะถูกมองเหมือนมีแรงดึงดูดในทิศจากตัวหุ่นไปยังพิกัดเป้าหมาย ยิ่งไกลเป้าหมายก็ยิ่งดูดแรง ส่วนสิ่งกีดขวางแต่ละชิ้นที่ตัวตรวจรู้จับได้จะสร้างแรงผลัก ยิ่งใกล้ ก็ยิ่งผลักแรง แรงลัพธ์ที่เกิดจากการบวกแรงดูดและแรงผลักทั้งหมดจะนำไปใช้ในการควบคุมการเคลื่อนที่ของหุ่น

[ref1]

Arkin, R. C. **AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents**, Plenary lecture and workshop presentation entitled "Just What is a Robot Architecture Anyway? Organizing Principles versus Turing Equivalency", Stanford, CA, March 1995.
http://www-static.cc.gatech.edu/ai/robot-lab/online-publications/stanford3.pdf

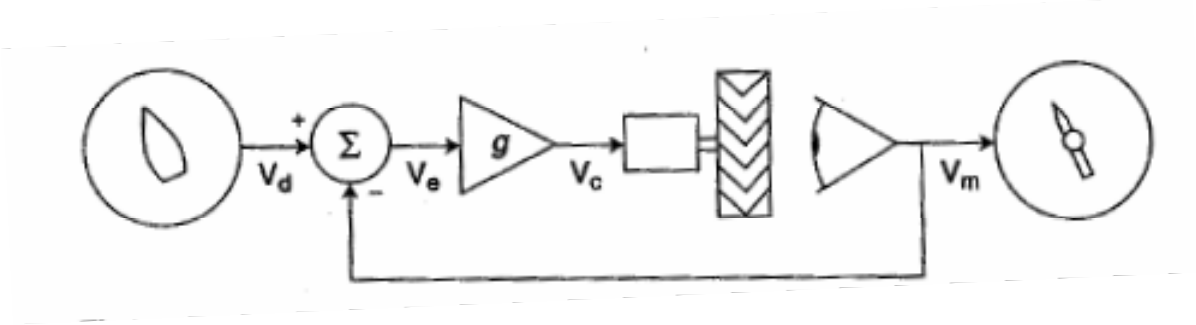
การควบคุมฉบับเตรียมอนุบาล

ความต้องการให้ทุกอย่างเป็นไปดั่งใจเป็นความทะเยอทะยานตามธรรมชาติของมนุษย์ เมื่ออะไรไม่เป็นตามแผน เราก็จะพยายามปรับเปลี่ยนวิธีการ เรื่องของการควบคุมก็เช่นกัน เป็นเรื่องที่ว่าด้วยการปรับอินพุท เพื่อให้เอาท์พุทเป็นไปตามต้องการ การควบคุมที่เราสนใจในที่นี้คือการควบคุมด้วยสัญญาณไฟฟ้า ตัวอย่างการควบคุมแบบง่าย เช่นพัดลม ที่มีปุ่มปรับเป็นเลข 1,2,3,4 เลขมากขึ้นก็ให้โวลเตจมากขึ้น และใบพัดก็หมุนเร็วขึ้น แต่การควบคุมพัดลมก็ไม่สนใจอุณหภูมิรอบตัวมัน เช่นกดไว้เบอร์ 2 เย็นกำลังพอดี แต่ถ้าฝนตกอุณหภูมิรอบๆ ลดลง พัดลมก็ยังเป่าแรงเท่าเดิม เราก็อาจจะว่าหนาวเกินไปได้ คราวนี้ลองดูการควบคุมเครื่องปรับอากาศ เรามีรีโมทตั้งอุณหภูมิ สมมุติเราตั้งไว้ 27 องศา เปิดไปแล้วอุณหภูมิห้องต่ำลงกว่า 27 องศา แอร์ก็จะตัด อากาศจะค่อยๆ ร้อนขึ้นๆ ถ้ามันร้อนกว่า 27 องศา แอร์ก็จะเปิดขึ้นมาอีก เพื่อให้อากาศเย็นขึ้น (จริงๆ ไม่ได้เปิด ปิดตรง 27 อาศาเป๊ะ ไม่งั้นก็จะเปิด-ปิด ถี่มากๆ จนแอร์พัง) จะเห็นว่าการควบคุมเครื่องปรับอากาศสนใจเอาผลลัพธ์จริงไปปรับการทำงาน เป็นการควบคุมที่เรียกว่า การควบคุมแบบมีป้อนกลับ (feedback control)



การควบคุมที่มีประสิทธิภาพเป็นเรื่องสำคัญในการควบคุมหุ่นยนต์ เราต้องการให้ล้อหุ่นยนต์หมุนด้วยความเร็วที่ต้องการ หากใช้การตั้งระดับโวลเตจคงที่สำหรับคุมมอเตอร์ล้อ ตอนขึ้นเนิน ต้องออกแรงมากขึ้น ความเร็วของล้อก็จะตก เช่นเดียวกันหากลงเนิน ก็จะไหลลง ความเร็วก็จะเพิ่มขึ้น เพื่อให้ได้ล้อหมุนด้วยความเร็วที่ต้องการ เราต้องมีวิธีอัตโนมัติที่จะปรับโวลเตจชดเชยกับงานที่ต้องทำ (load) เช่นเวลาขึ้นเนินก็ทำให้โวลเตจสูงขึ้น เป็นต้น วิธีควบคุมแบบมีป้อนกลับที่ง่ายสุด สำหรับงานแบบนี้ เรียกว่า การควบคุมแบบสัดส่วน (proportional

control) การควบคุมแบบนี้ เราจะต้องมีตัวตรวจรู้ เพื่ออ่านค่าความเร็วจริงของล้อ Vm แล้วเอาไปลบออกจากความเร็วที่ต้องการ Vd ได้เป็นค่า Ve = Vd-Vm จากนั้นเอาค่า Ve ไปคูณกับค่า gain g ได้เป็น Vc = g Ve ส่งไปคุมมอเตอร์ล้อ โดยคร่าวๆ พฤติกรรมของมอเตอร์นี้ก็คือว่าถ้าเราให้โวลเตจสูงขึ้น มอเตอร์ก็จะหมุนเร็วขึ้น เพื่อไปสู่ความเร็วคงที่ซึ่งสูงขึ้น แต่แน่นอนว่าการเปลี่ยนความเร็วของมอเตอร์ไม่สามารถเป็นไปอย่างฉับพลัน ต้องค่อยๆ เพิ่มขึ้นอย่างต่อเนื่อง เช่นเดียวกันหากเราให้โวลเตจต่ำลง ความเร็วก็จะค่อยๆ ลด เพื่อไปสู่ความเร็วคงที่ซึ่งต่ำลง ลองพิจารณาการทำงานของการควบคุมแบบสัดส่วน จะเห็นได้ว่า สมมุติว่าตอนนี้ Vc > 0 และความเร็วได้ตามต้องการ นั่นคือ Vm=Vd และทำให้ Ve=0 ส่งผลให้ Vc=0 และทำให้ความเร็วลดลง ความเร็วของมอเตอร์จะลดลง และเมื่อตรวจสอบความเร็ว จะได้ว่า Vm<Vd ทำให้ Ve>0 และ Vc เพิ่มขึ้น ทำให้ความเร็วเพิ่มขึ้น หากการตรวจสอบครั้งต่อมา Vm>Vd เราจะได้ Ve<0 และ Vc เป็นลบ ทำให้ความเร็วลดลง ความเร็วจะไม่มีทางคงที่ ณ Vd ที่ต้องการ แต่จะแกว่งอยู่รอบๆ จะแกว่งแคบแค่ไหนขึ้นกับความถี่ในการตรวจจบความเร็วเพื่อปรับค่า Vc ยิ่งตรวจจับถี่ก็จะสามารถแกว่งในช่วงที่แคบกว่า



หลายคนอาจมีความคิดว่าแทนที่จะส่ง Vc ให้มอเตอร์ ก็เอาเป็น Vc+Vd ไปเลยจะได้ไม่ต้องแกว่งตอนถึงความเร็วที่ต้องการ น่าสนใจ ลองวิเคราะห์ดูเองว่าจะเกิดอะไรขึ้น สิ่งที่ต้องคำนึงถึงในการวิเคราะห์นี้ก็คือว่า ค่า Vd, Ve, Vc และ Vm เป็นค่าโวลเตจทั้งหมด โดยหากเราป้อนโวลเตจ Vd ให้มอเตอร์โดยตรง จะทำให้มอเตอร์หมุนด้วยความเร็วที่แทนด้วยโวลเตจ Vd สำหรับค่า load ที่เหมาะสมเพียงค่าหนึ่งเท่านั้น

อีกเรื่องที่น่าคิดก็คือความถี่ในการตรวจสอบความเร็วเพื่อปรับค่า Vc หากความถี่นี้ไม่สูงพอจะเกิดอะไรขึ้น ลองดูสถานการณ์ง่ายๆ (ในรูปข้างล่าง) ในการควบคุมให้หุ่นยนต์หยุด ณ จุดที่ความเข้มของแสงเป็น Ld สมมุติว่าหุ่นยนต์วิเคราะห์ข้อมูลความเข้มแสง ณ เวลา t=0 แล้วได้ผลว่าต้องทำความเร็วให้หุ่นเคลื่อนไปข้างหน้าเพิ่มขึ้น แต่กว่าจะได้ผลสรุปนี้ ก็เป็นเวลา t=a+b+c เพราะต้องใช้เวลาในการประมวลผล ได้แก่การ แปลง A/D จากตัวตรวจรู้ การดึงข้อมูลดิจิทัลที่ได้เข้าประมวลผล และการคำนวณความเร็วใหม่ จากนั้นต้องส่งความเร็วใหม่ไปมอเตอร์ ซึ่งหุ่นจะเคลื่อนด้วยความเร็วใหม่นี้ ณ เวลา t=a+b+c+d+f ซึ่งมันก็เลยตำแหน่งเป้าหมายไปแล้ว สิ่งที่

ต้องคำนึงถึงก็คือ ผลสรุปที่ได้จากการคำนวณไม่สามารถทำให้เกิดผลได้ทันที แต่ต้องใช้เวลา ซึ่ง ณ เวลาที่เกิดผลนั้น ข้อมูลที่ได้ใช้คำนวณอาจล้าสมัยไปแล้ว ผลที่ได้ก็อาจไม่ตรงที่ต้องการ พูดง่ายสุดก็คือพยายาม update ด้วยความถี่สูงพอ จะได้ไม่เจอปัญหานี้ แต่หากทำไม่ได้ อาจต้องมีการใช้เทคนิคอื่นๆ ช่วย เช่นการทำนายอนาคตเพื่อเอาค่าจากการทำนายมาใช้ประกอบการคำนวณ

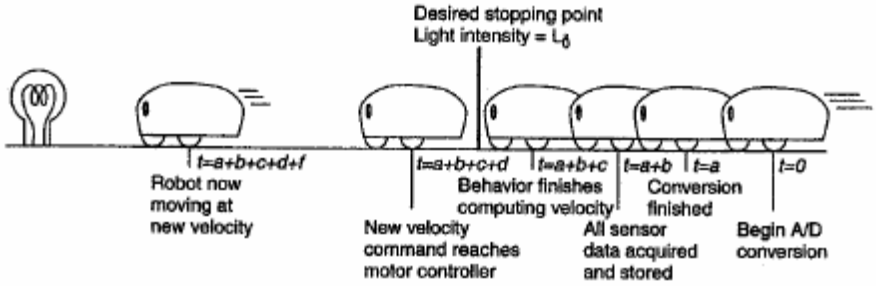


Figure 2.5

Latency in the system controlling the robot causes the robot to overshoot the desired position. At $t = 0$ with the robot moving at high velocity toward the light, the robot begins to acquire the sensory information on which it will base its next velocity command. After several intermediate computations at time $t = a + b + c + d + f$, the robot finally reaches the velocity that would have been appropriate at $t = 0$. By this time, however, the robot has passed the desired stopping position. The robot's velocity command will have the wrong sign for one more iteration of the control system.

การควบคุมแบบสัดส่วนเป็นกรณีพิเศษของการควบคุมที่นิยมมากที่สุด ซึ่งมีชื่อว่า PID (Proportional-Derivative-Integral Controller) การควบคุมแบบ PID นอกจากค่าความผิดพลาด e จะผ่านการคูณด้วยค่า proportional gain ให้ได้เป็น $g_P e(t)$ เหมือนการควบคุมแบบสัดส่วน ยังต้องนำไปรวมกับ $\int g_I e(t)dt$ และ $g_D \frac{de(t)}{dt}$

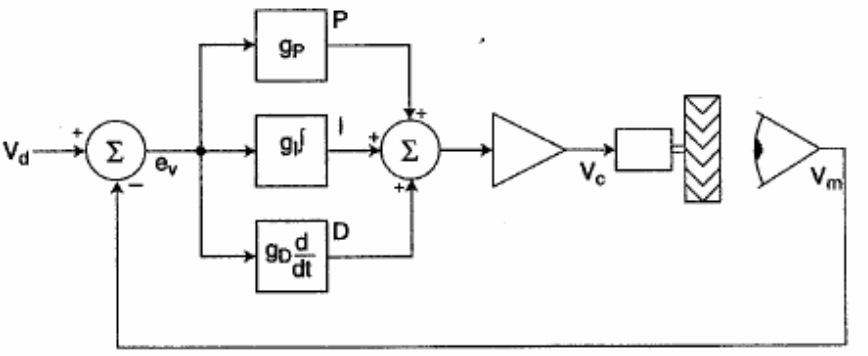
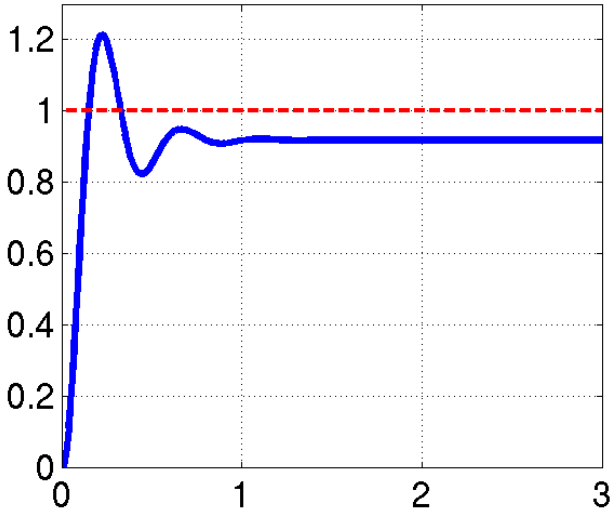


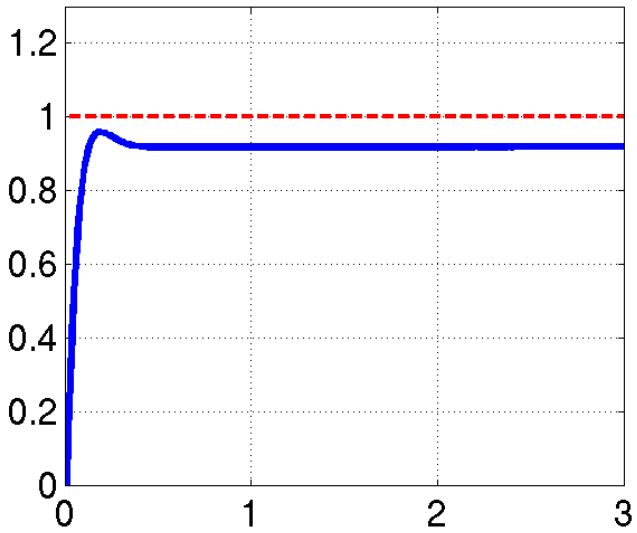
Figure 2.7

A general PID control loop includes a differential and an integral term as well as a proportional term. The D or differential branch looks at how quickly the error term is changing and modifies the error signal based on the rate of change. The I or integral branch integrates (adds up over time) the small offset uncorrected by the other branches and uses this to cancel out offset errors. Each branch has an associated gain: g_P, g_I, and g_D.

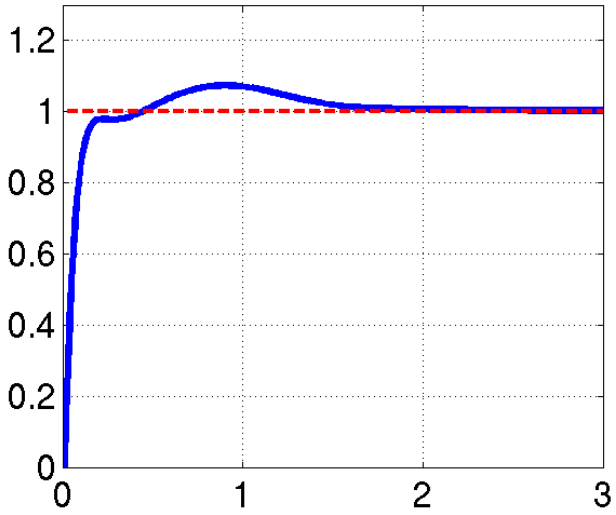
การวิเคราะห์การทำงานของ PID ต้องอาศัยพื้นฐาน differential equation แต่ในที่นี้จะขอกล่าวถึงเฉพาะการใช้งานเท่านั้น ซึ่งงานหลักของผู้ใช้ก็คือการปรับค่า g_P, g_I, g_D ให้เหมาะสม วิธีการที่ใช้นิยมใช้กันก็คือเริ่มจากการปรับค่า g_P โดยให้ g_I, g_D เป็นศูนย์ก่อน โดยค่อยๆ ปรับค่า g_P และสังเกตว่าผลลัพธ์จะปรับตัวเข้าค่าที่ต้องการ ยิ่งให้ g_P มากก็ปรับสู่ค่าที่ต้องการอย่างรวดเร็ว แต่หากมากเกินไป อาจทำให้ปรับตัวเลยค่าที่ต้องการ ซึ่งอาการนี้เรียกว่า overshoot เหมือนขับรถเลยไปเบรกไม่ทัน สิ่งที่เราต้องการคือให้ผลลัพธ์ปรับเข้าค่าที่ต้องการอย่างรวดเร็ว และไม่ต้องการให้เกิด overshoot



การแก้ overshoot ทำได้โดยพยายามลด g_P หรือไม่ก็เพิ่มส่วน derivative เข้ามา โดยต้องมีการปรับเพิ่มค่า g_D หน้าที่ของมันก็เหมือนกับการแตะเบรคเวลา overshoot เป็นตัวระงับการเปลี่ยนแปลงแบบฉับพลัน ทำให้เข้าสู่ค่าที่ต้องการได้เร็วขึ้น



อย่างไรก็ตามหลังการปรับทั้ง g_P, g_D ก็อาจยังมีส่วน error ที่เอาไม่ออก ถึงแม้ผลลัพธ์จะคงตัวแล้วก็ตาม เราเรียก error แบบนี้ว่า steady state error วิธีแก้อาการนี้คือการเพิ่มในส่วนของ g_I การเพิ่มในส่วนนี้จะทำให้เกิดการหักล้างกับ error สะสม แต่อาจทำให้เวลาที่ใช้กว่าสัญญาณจะคงตัวนานขึ้น



สรุปผลกระทบจากการปรับเพิ่มของแต่ละพารามิเตอร์

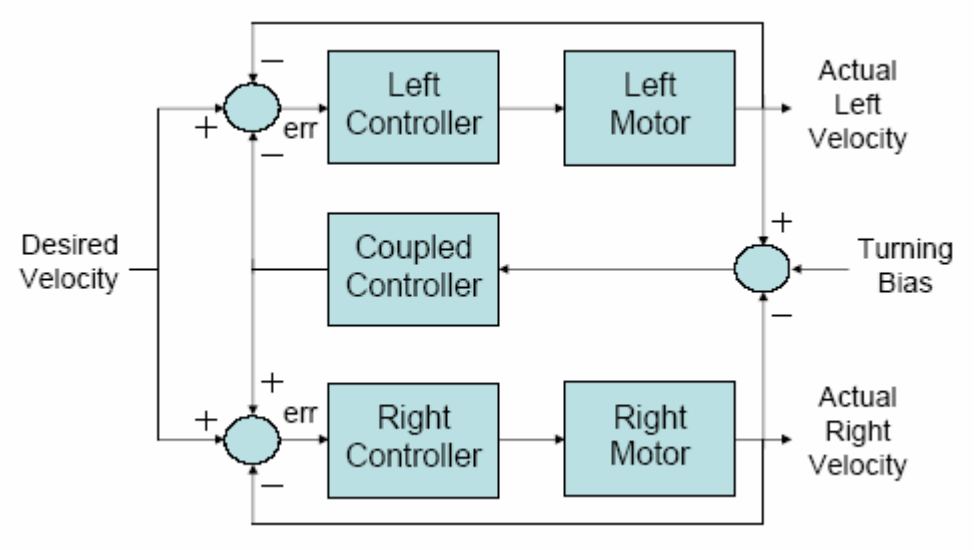
	Rise time	Overshoot	S-S error	Settling time
g_P	ลด	เพิ่ม	ลด	กระทบน้อย
g_I	ลด	เพิ่ม	กำจัด	เพิ่ม
g_D	กระทบน้อย	ลด	กระทบน้อย	ลด

เพื่อให้ได้เห็นประสบการณ์การปรับค่าพารามิเตอร์ของ PID ขอให้ทดลองปรับพารามิเตอร์ด้วย applet ใน web

เราสามารถเขียนโปรแกรมควบคุมแบบ PID ด้วยภาษา C ได้ดังต่อไปนี้ โดยเราต้องกำหนดค่าพารามิเตอร์ g_P, g_I, g_D ให้ตัวแปร m_kp, m_ki, m_kd โดยผ่าน function PID_Initialize นอกจากนี้ก็ต้องกำหนดค่าต่ำสุดของ error ที่นับสำหรับการอินติเกรทในตำแปร m_error_thresh และระยะเวลา 1 time step ในตัวแปร m_h ค่าระยะเวลานี้จะถูกใช้ในการหา derivative และ integrate ด้วยวิธีทาง numerical

```
#include <math.h>

/* Select 'double' or 'float' here: */
typedef double real;
void PID_Initialize(real kp, real ki, real kd,
    real error_thresh, real step_time);
real PID_Update(real error);
static int m_started;
static real m_kp, m_ki, m_kd, m_h, m_inv_h, m_prev_error,
        m_error_thresh, m_integral;

void PID_Initialize(real kp, real ki,
    real kd, real error_thresh, real step_time)
{
    /* Initialize controller parameters */
    m_kp = kp;
    m_ki = ki;
    m_kd = kd;
    m_error_thresh = error_thresh;

    /* Controller step time and its inverse */
    m_h = step_time;
    m_inv_h = 1 / step_time;

    /* Initialize integral and derivative calculations */
    m_integral = 0;
    m_started = 0;
}

real PID_Update(real error)
{
    real q, deriv;
    /* Set q to 1 if the error magnitude is below
       the threshold and 0 otherwise */
    if (fabs(error) < m_error_thresh)

        q = 1;
    else
        q = 0;
    /* Update the error integral */
    m_integral += m_h*q*error;

    /* Compute the error derivative */
    if (!m_started)
    {
        m_started = 1;
        deriv = 0;
    }
    else
        deriv = (error - m_prev_error) * m_inv_h;
    m_prev_error = error;
    /* Return the PID controller actuator command */
    return m_kp*error + m_ki*m_integral + m_kd*deriv;

}
```

แผนผังการใช้ตัวควบคุม PID ในการคุมมอเตอร์ล้อทั้งสองของหุ่นยนต์แบบ differential drive ให้หมุนอย่างสอดคล้องกัน แต่ละมอเตอร์จะมีตัวควบคุมหนึ่งตัว และสามารถสั่งเลี้ยวโดยใส่สัญญาณ Turning Bias ผ่านตัวควบคุมตัวที่สาม



นอกจากเราสามารถใช้ PID ในการควบคุมความเร็วมอเตอร์ เรายังสามารถใช้ PID ในการควบคุมอื่นๆ ซึ่งใช้ข้อมูลจากอุปกรณ์ต่างๆ เช่น กล้อง