Robot Design

Summer course 2007

BEST@LTH

Jacek Malec

Acknowledgements

Jacek Malec has prepared his lectures basing, among other material, on:

the collection of eight lectures prepared for the Autumn 2000 EE462 class, "Principles of Mobile Robots," at the University of Washington using Fred Martin's pre-publication text, Robotic Explorations: A Hands-on Introduction to Engineering, Prentice Hall, 2001.

The PowerPoint slides were created by Dr. Linda Bushnell, bushnell@ee.washington.edu

Please see the EE462 course web site for more information on the syllabus, laboratory assignments, homework assignments, and links:
http://www.ee.washington.edu/class/462/bushnell/

Outline

- Technology: LEGO (mechanics), RCX (hardware), NQC (software)
- Simple reactivity - feedback loop
- reactivity vs. deliberation - algorithmic control, world modelling, knowledge-based control
- Agent architectures, subsumption
- Software for embedded systems

Braitenberg Vehicles

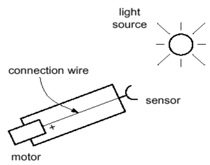
•Neuro-biologist Valentino Braitenberg, *Vehicles: Experiments into Synthetic Psychology* (1984). "how sentient creatures might have evolved from simpler organisms"

•Vehicle 1: 1 Motor/1 Sensor

–Wire connects sensor to motor

–Sensor generates a signal proportional to the strength of light

–When it "sees" a light source, it starts moving in straight line

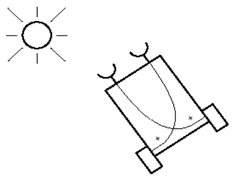


Braitenberg Vehicles

•Vehicle 2b: 2 Motors/2 Sensors

–Turns towards light source

–Reduces difference between heading and brightest source of light (negative feedback)



Feedback

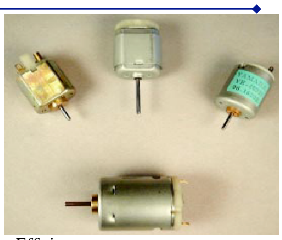
Automatic control
Cybernetics

Early example:
James Watt's governor
for steam engine speed regulation

DC Motors

Direct Current (DC) Motors:

- Small, cheap, reasonably efficient, easy to use, ideal for small robotic applications
- Converts electrical energy into mechanical energy
- How do they work?
  - By running electrical current through loops of wires mounted on rotating shaft (*armature*)
  - When current is flowing, loops of wire generate a magnetic field, which reacts against the magnetic fields of permanent magnets positioned around the wire loops
  - These magnetic fields push against one another and the armature turns



- Efficiency
  - Various limitations, including mechanical friction, cause some electrical energy to be wasted as heat
  - Toy motors: efficiencies of 50%
  - Industrial-grade motors: 90%

DC Motors

Properties:

- **Operating Voltage**
  - Recommended voltage for powering the motor
  - Most motors will run fine at lower voltages, though they will be less powerful
  - Can operate at higher voltages at expense of operating life
- **Operating Current**
  - When provided with a constant voltage, a motor draws current proportional to how much work it is doing
  - When there is no resistance to its motion, the motor draws the least amount of current; when there is so much resistance as to cause the motor to stall, it draws the maximal amount of current
  - *Stall current:* the maximum amount of operating current that a motor can draw at its specified voltage

DC Motors

Properties:

- **Torque**
  - Rotational force that a motor can deliver at a certain distance from the shaft
    - The more current through a motor, the more torque at the motor's shaft
  - Direct consequence of the electromagnetic reaction between the loops of wire in the motor's armature and the permanent magnets surrounding them
  - Strength of magnetic field generated in loops of wire is directly proportional to amount of current flowing through them; torque produced on motor's shaft is a result of interaction between these two magnetic fields
  - Often a motor will be rated by its *stall torque*, the amount of rotational force produced when the motor is stalled at its recommended operating voltage, drawing the maximal stall current at this voltage
  - Typical torque units: *kilogram-meter*; e.g., 0.01 kgm. torque means motor can pull weight of 1 kg up through a pulley 1cm away from the shaft.

DC Motors

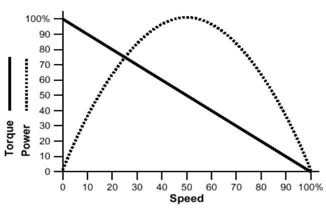
Properties:

- **Power**
  - Product of the output shaft's *rotational velocity* and torque
  - Output Power Zero
    - Case 1: Torque is zero
      - Motor is spinning freely with no load on the shaft
      - Rotational velocity is at its highest, but the torque is zero—it's not driving any mechanism (Actually, the motor is doing some work to overcome internal friction, but that is of no value as output power)
    - Case 2: Rotational Velocity is zero
      - Motor is stalled, it is producing its maximal torque
      - Rotational velocity is zero
  - In between two extremes, output power has a characteristic parabolic relationship

DC Motors

Motor Speed vs. Torque, Power:

- **Solid line** shows the relationship between motor speed and torque
  - At the right of the graph, the speed is greatest (100%) and the torque is zero; this represents the case where the motor shaft is spinning freely but doing no actual work
  - At the left of the graph, the speed is zero but the torque is at its maximum; this represents the case where the shaft is stalled because of too much load
- **Dashed line** shows the power output, which is the product of speed and torque
  - It is the highest in the middle of the motor's performance range, when both speed and torque are produced

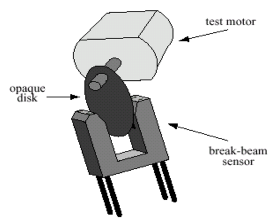


Idealized Graph

DC Motors

Measuring Motor's Top Speed in RPM:

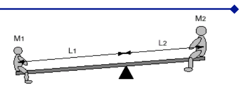
- Opaque disk (light-weight) is mounted directly on the motor shaft
- *Break-beam opto-sensor* is positioned such that as the disk rotates, it interrupts the sensor's light beam once per revolution
- For counting the transitions on the sensor, use *pulse accumulator input* (PAI = sensor input #9), which counts pulses on a particular digital input pin with hardware ancillary to the 6811 core (allows very fast rate, transparent to the rest of the processor's functioning)
- Most DC motors have unloaded speeds in the range of 3,000 to 9,000 revolutions per minute (RPM), which translates to between 50 and 150 revolutions per second. This is slow enough that a regular 68HC11 analog input could be used, but it is possible that Interactive C would not be able to keep up with this rate.



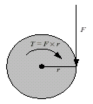
Gearing

- DC motors are high-speed, low-torque devices
- All mechanisms in robots, including drive trains and actuators, require more torque and less speed
- **Gears** are used to trade-off high speed of the motor for more torque
- Torque, or rotational force, generated at the center of a gear:

T = F x r



Downward force is equal to weight times their distance from the fulcrum. Lighter people can displace heavier people simply by increasing their distance from the fulcrum.



The torque t—or, turning force—is the product of a force F applied perpendicularly at a radius r.

Gearing

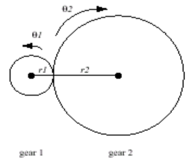
Meshing Gears

- When two gears of unequal sizes are meshed together, their respective radii determine the translation of torque from the driving gear to the driven one
- This mechanical advantage is easiest understood from a "conservation of work" point of view

W = F x d

W = T x θ

- Neglecting losses due to friction, no work is lost or gained when one gear turns another
- Example: Gear 1's radius is one-third that of Gear 2. Their circumferences are also in a 3:1 ratio, so it take three turns of the small gear to produce one turn of the larger gear. Ratio of resulting torques is also 3:1.



Gear 1 with radius r1 turns an angular distance of θ1 while **Gear 2** with radius r2 turns an angular distance of θ2.

Ratio of gear sizes determines ratio of resulting torques

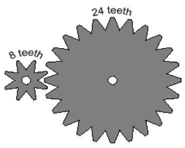
Gearing

Gear Reduction

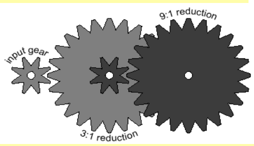
- Small gear driving a larger one:
  - **torque increases**
  - **speed decreases**

Exercise: calculate effective gear ratio of *HandyBug*'s drive train

- 3 to 1 Gear Reduction
  - Power applied to 8-tooth gear results in 1/3 reduction in speed an 3 times increase in torque at 24-tooth gear
- 9 to 1 Gear Reduction
  - By putting two 3:1 gear reductions in series—or "ganging" them—a 9:1 gear reduction is created
  - The effect of each pair of reductions is multiplied to achieve the overall reduction
  - Key to achieving useful power from a DC motor
  - With this gear reduction, the high speed and low torque is transformed into usable speeds and powerful torques



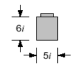
3 turns of left gear (8 teeth) to cause 1 turn of right gear (24 teeth)



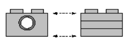
8-tooth gear on left; 24-tooth gear on right

LEGO Design

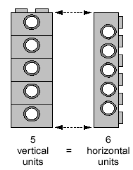
Structure



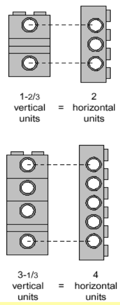
Unit LEGO brick
i is a conversion factor between "LEGO lengths" and standard units
6/5 height full-size brick



Three of the **thin LEGO plates** are equal in height to the unit brick
2/5 height thin plate



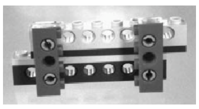
Stack of Five LEGO Bricks = Six-Long LEGO Beam



Two-Unit and Four-Unit Vertical LEGO Spacings

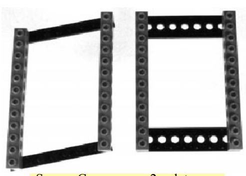
LEGO Design

Structure



Sturdy LEGO construction

Full Height Bricks	One-Third Height Plates	Horizontal Units
1	2	2
3	1	4
5	0	6
6	2	8

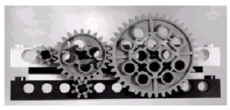


Square Corners: use 2x plates rather than 1x ones

Black peg is slightly larger; fits snugly
Gray peg rotates freely

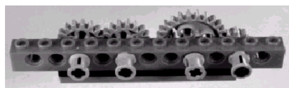
LEGO Design

Gearing

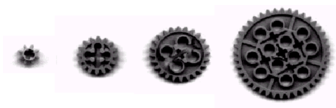


The 8–tooth, 24–tooth, and 40– tooth round gears all mesh properly along a horizontal beam because they have **"half unit" radii.** The 8– and 24– tooth gears are meshed horizontally at two units, and vertically.

Number of Teeth	Radius in Horizontal Units
8	1/2
16	1
24	1–1/2
40	2–1/2

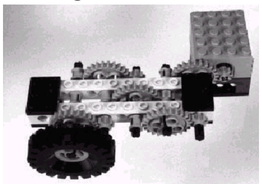


The 16–tooth gear has a radius of **1 LEGO unit,** so two of them mesh properly together at a spacing of two units. Since an 8– and 24– tooth gear also mesh at two-unit spacing, these respective pairs of gears **can be swapped** for one another in an existing geartrain.

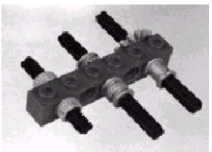


8–tooth 16–tooth 24–tooth 40–tooth

Gearing

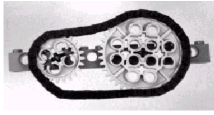


A five-stage reduction using 8– and 24–tooth gears creates a **243-to-1 reduction** in this sample LEGO geartrain. Note the need for three parallel planes of motion to prevent the gears from interfering with one another. Four 2x3 LEGO plates are used to hold the beams square and keep the axles from binding.

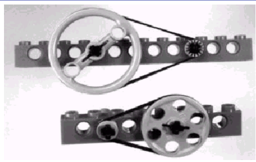


- Standard 1-LEGO-long **stop bush** (upper axle, front) is not the only part that can act as a **bushing** (axle holder)
- Small **pulley wheel** (middle axle) acts as a half-sized spacer—it also grabs tighter than the full bush
- **Bevel gear** (upper axle, back) makes a great bushing
- **Nut-and-bolt parts** (lower axle) can be used to make a tight connection

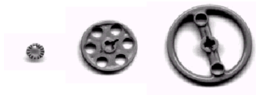
Chain Links and Pulleys



Chain links can be an effective way to deliver large amounts of torque to a final drive, while providing a gear reduction if needed. Chain link works best at the slower stages of gearing, and with a somewhat slack link-age. Use the larger gears—the 8–tooth one won't work very well.

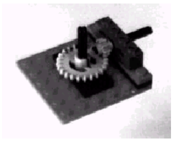


There are three sizes of **pulley wheels**:
- Tiny one, which doubles as a **stop bush**
- Medium-sized one, which doubles as a **tire hub**
- Large-sized one, which is sometimes used as a **steering wheel** in official LEGO plans



small pulley medium pulley large pulley

Crown and Bevel Gears



24–tooth crown gear

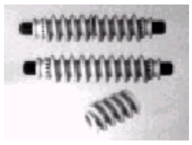
bevel gear

The 8–tooth gear, in conjunction with the 24-tooth **crown gear**, is used to **change the axis of rotation** in a gear train. In this instance, the configuration provides for a **vertical shaft output.** Horizontal output also possible.

The **bevel gears** are used to **change the angle of rotation** of shafts in a gear train with a 1:1 ratio. In this case, they are used to effect a change in the **horizontal plane.**

Worm Gear

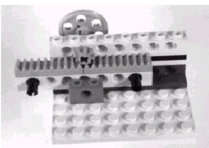
The **worm gear** is valuable because it acts as a **gear with one tooth**: each revolution of the worm gear advances the round gear it's driving by just one tooth. So the worm gear meshed with a 24-tooth gear yields a **24:1 reduction**. The worm gear, however, loses a lot of power to friction, so it may not be suitable for high performance, main drive applications.



- **Bottom** is the basic worm gear, two horizontal LEGO units in length
- **Top** is an unsuccessful attempt to put two worm gears on the same shaft
- **Middle** is the successful attempt

When placing multiple worm gears on a shaft, the trick is to try all four possible orientations to find the one that works.

Gear Rack



The gear driving the **gear rack** is often referred to as the **"pinion,"** as in "rack-and-pinion steering," which **uses the transverse motion of the gear rack to orient wheels**. The 8–tooth gear is a good candidate to drive the rack be-cause of the gear reduction it achieves—one revolution of the gear moves the rack by eight teeth.

Geartrain Design Tips

- Work **backward from the final drive**, rather than forward from the motor
  - Usually there is a fair bit of flexibility about where the motor is ultimately mounted, but much less in the placement of drive wheels or leg joints
  - Start by mounting the axle shaft that will carry the final drive, put a wheel and gear on it, and start working backward, adding gearing until there is enough, and finally mount the motor in a convenient spot
- Do not forget about the **role of the tire** in determining the relationship between the rotational speed of the final drive axle and the linear speed that is achieved
  - Small tires act as gear reductions with respect to large tires, and this may have an effect on how much gear reduction is necessary
- If **geartrain performing badly**
  - Make sure the stop bushes are not squeezing too hard—there should be some room for the axles to shift back and forth in their mounts
  - Check that all beams holding the axles are squarely locked together
- To **test a geartrain**, try driving it **backward**
  - If your geartrain can be readily back-driven, it is performing well

LEGO Design

LEGO Clichés (from Fred Martin)

On occasion it is necessary to **lock a beam to an axle**. This figure shows how to use a medium **pulley wheel**, which rigidly locks to an axle, to hold the beam in place.

The special **"gear mounter"** piece is an axle on one side and a loose connector peg on the other. It can be used to mount gears used as idlers in a gear train — used simply to transmit motion or to reverse the direction of rotation.

This configuration of parts can be used as a **compact axle joiner.** LEGO now produces a part designed for this purpose, but in lieu of that part, this is a useful trick.

LEGO Design

LEGO Clichés (from Fred Martin)

In order to **build outward from a vertical wall** of axle holes, a smaller beam may be mounted with its top studs in the holes of the beam wall.

The recommended way to build outward from a beam wall is to use the **connector-peg-with-stud piece,** which is a loose-style connector peg on one end and a top stud on the other.

The **full-size stop bush** can be used in one orientation to **hold an axle through a plate hole** so that the axle can freely rotate.

LEGO Design

LEGO Clichés (from Fred Martin)

By using the stop bush to hold an axle in place between two plates, a **vertical axle mount** can easily be created. Depending on the orientation of the stop bush, it can be made to either lock the axle in place or allow it to rotate freely.

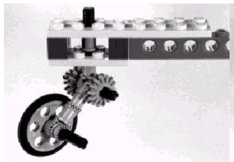
In the other orientation, the stop bush locks between four top studs, perfectly centered over the axle holes in flat plates. This allows the stop bush to **lock a plate to an axle.**

The **"toggle joint"** can be used to lock two axles at a variety of odd angles. The short axle running through the two toggle joints is equipped with stop bushes on either end to hold the joint together.

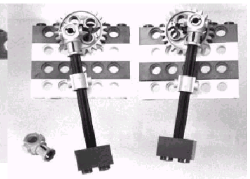
LEGO Design

LEGO Clichés (from Fred Martin)

Here the **toggle joint** is used to connect two axles at right angles. The small pulley wheel is deployed on the axle that runs through the toggle joint to either **lock the axle or allow it to rotate.**



Several clichés are used to construct this **caster wheel**.

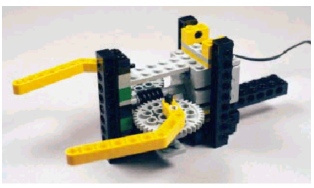


The **"piston rod"** part is used twice in each mechanism to create a **LEGO leg**. By using a chain drive or gear linkage to lock legs in sync, a multi-legged creature can be designed.

LEGO Design

LEGO Clichés (from Fred Martin)

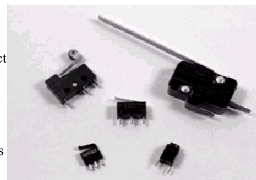
Robot Gripper Using Gear Rack



Robot Gripper Using Worm Gear

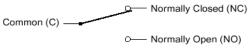
Switch Sensors

- **Contact (touch) Sensing**
  - Switch sensors can be used to indicate when a mechanism has made physical contact with another object
  - e.g., it can trigger when a robot's body runs into a wall, or when a robot's gripper closes around a cube
- **Limit Sensing**
  - Related to simple contact sensing, a limit sensor detects when a mechanism has moved to the end of its range of travel, signaling to the control program that the motor should be turned off
- **Shaft Encoding**
  - As with past instances of shaft encoding, an axle may be fitted with a contact switch that clicks once per revolution. Software that counts the clicks can then determine the amount and speed of the axle's rotation.



Various Switches

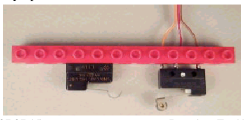
Switch Sensors

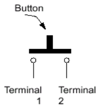


Microswitches typically have three terminals:

- "NO" (normally open)
- "NC" (normally closed)
- "C" (common)

Common terminal may connect to either of the other two terminals, depending on whether or not the switch is pressed. In the relaxed, un-pressed state, the common terminal is connected to the normally closed contact; when pressed, the common terminal moves to the normally open contact.



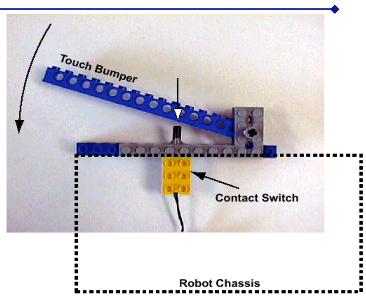


A **pushbutton switch** is simpler:

- Normally open pushbutton: when the switch is pressed, the two contacts are connected.
- Normally closed pushbuttons also exist, but these are less common.

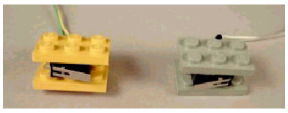
Switch Sensors

Switch Sensor Applications



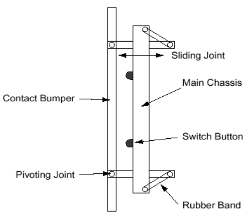
Design for a Simple Touch Bumper

Left- and Right-Hand Switch Construction

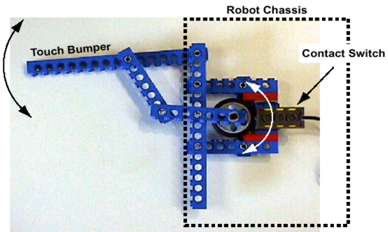


Switch Sensors

Switch Sensor Applications



HandyBug Bumper Design
- rotational and sliding pivot points allow the bumper to react to pressure from any forward direction



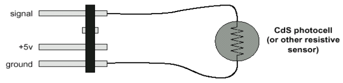
Design for Bi-Directional Touch Bumper
- can detect pressure from front or behind
- movement in either direction pushes levered arm away from contact sensor
- rubber bands pull arm back onto switch when pressure is released

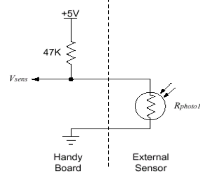
Light Sensor Circuits

Single Photocell Circuit

- Photocell element is connected to the circuit ground and the HB's sensor input line via a voltage divider circuit
- V_{sens} , resulting sensor voltage, varies as to the ratio between 47KΩ and R_{photo}
  - When the photocell resistance is small (brightly illuminated), the V_{sens} ~= 0v
  - When the photocell resistance is large (dark), V_{sens} ~= +5 v — **"dark sensor"**
  - Continuously varying range between extremes
- Sensor will report small values when brightly illuminated and large values in the dark
- May invert the sense of the readings from the HB's analog ports:

```
int light(int port) {return 255 - analog(port);}
```

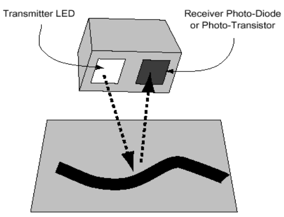




Photocell Voltage Divider Circuit

Reflective Optosensors

- Active Sensor - includes own source of quantity being detected
- Commercial ones use infrared light; include filter to pass infrared wavelengths while blocking visible light
- Reflective optosensor includes a source of light (emitter LED) and a light detector (photodiode or phototransistor)
  - Arranged in a package so that light from emitter LED bounces off of an external object (e.g., the black line on a surface) and is reflected into the detector
  - Depending on the reflectivity of the surface, more or less of the transmitted light is reflected into the detector
  - Quantity of light is reported by the sensor



Reflective Optosensors

Applications

Object detection. Reflectance sensors may be used to measure the presence of an object in the sensor's field of view. In addition to simply detecting the presence of the object, the data from a reflectance sensor may be used to **indicate the object's distance** from the sensor. These reading are dependent on the reflectivity of the object, among other things—a highly reflective object that is farther away may yield a signal as strong as a less reflective object that is closer.

Surface feature detection. Reflective optosensors are great for detecting features painted, taped, or otherwise marked onto the floor. **Line-following** using a reflective sensor is a typical robot activity.

Wall tracking. Related the object detection category, this application treats the wall as a continuous obstacle and uses the reflective sensor to indicate **distance from the wall**.

Rotational shaft encoding. Using a pie-shaped encoder wheel, the reflectance sensor can measure the rotation of a shaft (**angular position and velocity**).

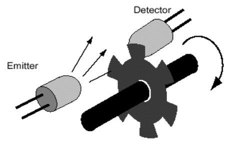
Barcode decoding. Reflectance sensors can be used to decode information from barcode markers placed in the robot's environment.

Shaft Encoding

• Use Break-Beam Sensors

• Shaft encoder measures the angular rotation of an axle, reporting position and/or velocity information

• Example: **speedometer**, which reports how fast the wheels are turning; **odometer**, which keeps track of the number of total rotations



Single-Disk Shaft Encoder
A perforated disk is mounted on the shaft and placed between the emitter–detector pair. As the shaft rotates, the holes in the disk chop the light beam. Hardware and software connected to the detector keeps track of these light pulses, thereby monitoring the rotation of the shaft.

Shaft Encoding

Measuring Velocity

• Driver routines measure rotational velocity as well as position

– Subtract difference in the position readings after an interval of time has elapsed

• Velocity readings can be useful for a variety of purposes

– Robot that has an **un-powered trailer wheel** with a shaft encoder can easily tell whether it is moving or not by looking at encoder activity on the trailer wheel. If the robot is moving, the trailer wheel will be dragged along and will have a non-zero velocity. If the robot is stuck, whether or not its main drive wheels are turning, the trailer wheel will be still.

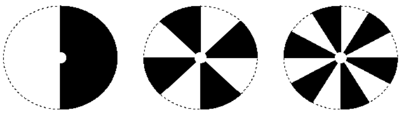
• Velocity information can be **combined with position information** to perform tasks like causing a robot to drive in the straight line, or rotate a certain number of degrees. These tasks are inherently unreliable because of mechanical factors like slippage of robot wheels on the floor and backlash in geartrains, but to a limited extent they can be performed with appropriate feedback from shaft encoders.

Shaft Encoding

Reflective Optosensors as Shaft Encoders

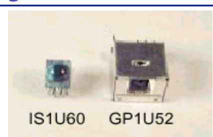
• It's possible to build shaft encoders by using a reflective optosensor to detect black and whi markings on an encoder wheel

• Wheels can be used with any of the reflective optosensor devices, as long as the beam of light they generate is small enough to fit within the black and white pie-shaped markings



Infrared Sensing

- Simple IR sensing:
  - Reflectivity sensing or break-beam sensing
  - Exactly analogous to using a light bulb, candle flame, or other constant light source with a visible-light photocell sensor
  - Sensor simply reports the amount of overall illumination, including both ambient lighting and the light from light source
- Advantage over resistive photocells:
  - Quicker to respond to light changes, so they are well-suited to the break-beam shaft encoding application
  - More sensitive, so with proper shielding from ambient light sources, can detect small changes in lighting levels.



Sharp Demodulators ($3)

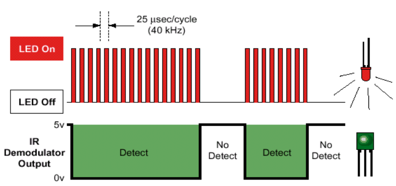
More powerful way to use infrared sensing:

- By rapidly **turning on and off the source** of light, the source of light can be easily picked up from varying background illumination—even if the actual amount of modulated light is very small
- Great insensitivity to background ambient lighting can be accomplished
- This is how tv remote controls work; infrared LEDs in the remote control transmit rapid flashes of light, which are decoded by a device in the tv

Infrared Sensing

Modulation and Demodulation

- Basic principle: by flashing a light source at a particular frequency (**modulation**), the flashes of light at that same frequency can be detected (**demodulation**), even if they are very weak with respect to overall lighting conditions
- Demodulator is tuned to a specific frequency of light flashes
  - Commercial IR demodulators range 32 - 45 KHz; high enough to avoid interference effects from common indoor lighting sources, like florescent lights
- Note negative true logic
- In practice, it takes 5-10 cycles for demodulation



Idealized Response of Infrared Demodulator
The upper graph indicates an infrared LED being turned on in two successive bursts. Each burst consists of a number of very rapid on-off pulses of light. The lower graph shows the output from the IR detector. During the rapid on-off bursts, the demodulator indicates "detection"; in between the bursts, the demodulator sees no IR activity, and indicates "no detection."

Infrared Sensing

Proximity Sensing

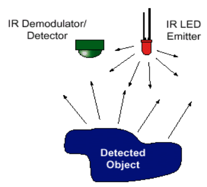
• Using the simple modulated output of an IR LED and an IR demodulator, it's possible to build an effective **proximity sensor**

• Light from the IR emitter is reflected back into detector by a nearby object, indicating whether an object is present (just like the simple (not modulated) reflectance sensors)

• LED emitter and detector are pointed in the same direction, so that when an object enters the proximity of the emitter-detector pair, light from the emitter is reflected off of the object and into the detector

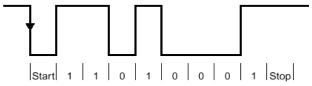
• This kind of simple **true-false proximity sensing** is an ideal application for modulated/demodulated IR light sensing

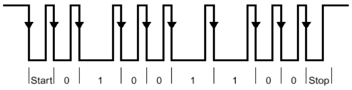
• Compared to simple reflected light magnitude sensing, modulated light is far less susceptible to environmental variables like amount of ambient light and the reflectivity of different objects



Infrared Communications

Serial Data Transmission Methods





Bit Frames:
• Each bit takes the same amount of time to transmit
• Synchronization is based on the falling edge of the Start bit; after that, following bits are determined by sampling the signal in the middle of the time period when the bit is valid (i.e., the bit frame)
• Method is good when the waveform can be **reliably transmitted** across a wire or other communications medium
• Used for standard computer/modem communication

Bit Intervals:
• **Amount of time between falling edges** determines whether a bit is 0/1
  – 0 represented by short interval
  – 1 represented by longer interval
• There is a short interval at the beginning to act as a start of frame, and a transition at the end to allow the last bit to be specified
• This method is good when it is **difficult to control the exact shape of the waveform** across the communications path
• Ideal for IR modulation/demodulation

Sensor Data Processing

• A big part of getting robot programs to function as intended lies in the interpretation of sensor data.

• If a robot's sensors not are performing or responding to the world as expected, it will be very difficult to have the robot react properly.

• In this section, we will explore a set of issues relating to the interpretation of sensor data, including

  – sensor **calibration** techniques

  – sensor **data filtering** techniques

Reference Activity: **Line Following**

• HandyBug with one downward-facing reflectance sensor

• Robot waddles back and forth across line, switching direction each time it has completely crossed over

• How do sensor functions work?

```
void line_follow() {
     while (1) {
          waddle_left();
          waituntil_on_the_line();
          waituntil_off_the_line();
          waddle_right();
          waituntil_on_the_line();
          waituntil_off_the_line();
     }
}
```


Sensor Data Processing

Fixed Thresholding

• Simplest, effective way to interpret sensor values is with *fixed thresholding*

• Sensor reading is compared with a **setpoint** value. If the reading is less than the setpoint, then the robot is assumed to be in "state A" (e.g., "on the line"); if the reading is greater than the setpoint, then the robot is in "state B" ("off the line").

• Process converts a continuous sensor reading—like a light level—to a digital state, much like a touch sensor is either pressed or not.

• **Line-following:** suppose the downward-facing reflective light sensor yields a reading of about 10 when aimed at the floor, and 50 when aimed at the line. It would then make sense to choose the **midpoint value of 30** as the setpoint for determining if the robot is on the line or not.

Parameterized Fixed Thresholding

• What if the setpoint value needs to change under different operating conditions?

• **Line Following:** setpoint value is hard-coded into two different routines—an approach that clearly does not scale as the program complexity increases.

• Better way: break out threshold setpoints as named variables or constants, and then refer to them by name in the actual routines

• When the setpoint needs to be changed, there is one clearly specified point in the program for this to be done

```
int LINE_SETPOINT= 30;
```


Sensor Data Processing

Thresholding with Hysteresis

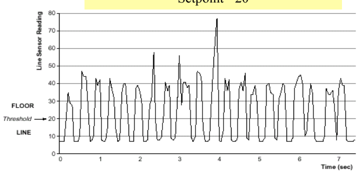
Line Following performance run :
Setpoint =20

• Sensor data is not extremely reliable

• **Line-following:** variances in ambient lightin and surface texture of the floor can easily creat unexpected and undesired glitches in sensor readings.

  – Bump on floor may spike the readings

  – Shiny spots on line may reflect as well as the floor, dropping the sensor reading up into the range of the floor



• Solution: **two setpoints** can be used

  – Imposes **hysteresis** on the interpretation of sensor values, i.e., prior state of system(on/off line) affects system's movement into a new state

```
int LINE_SETPOINT= 35;
int FLOOR_SETPOINT= 10;
void waituntil_on_the_line() {
     while (line_sensor() < LINE_SETPOINT);
}
void waituntil_off_the_line() {
     while (line_sensor() > FLOOR_SETPOINT);
}
```


Sensor Data Processing

Calibration by Demonstration

• Install manual calibration routines

• Robot is physically positioned over the line and floor and a threshold setpoint is captured

• **Calibrate ()** guides process of setting threshold setpoints for line/floor

• Huge improvement over fixed and hard-coded calibration methods

• Declare setpoint variables as **persistent** and use calibration routine

```
int LINE_SETPOINT= 100;
int FLOOR_SETPOINT= 100;
void main() {
     calibrate();
     line_follow();
}
void calibrate() {
     int new;
     while (!start_button()) {
         new= line_sensor();
         printf("Line: old=%d new=%d\n", LINE_SETPOINT,new);
         msleep(50L);
     }
     LINE_SETPOINT= new; /* accept new value */
     beep(); while (start_button()); // debounce button press
while (!start_button()) {
         new= line_sensor();
         printf("Floor: old=%d new=%d\n",FLOOR_SETPOINT,new);
         msleep(50L);
     }
     FLOOR_SETPOINT= new; /* accept new value */
     beep(); while (start_button()); // debounce button press
}
```


Sensor Data Processing

Sensor Histories

• Technique whereby sensor thresholds may be determined automatically, and can dynamically adjust to changing operating conditions. This and related methods have the opportunity to make robot behavior much more robust in the face of the variability and uncertainty of the real world.

• **Line Following:** Add code to automatically calculate a midpoint between the on-going maximum and minimum values, and use this midpoint as the line threshold.

  – Does not work well in practice: maximum values recorded as robot passes over line are much higher than typical line values. Robot does not see line. Routine fails.

• **Problem:** just having minimum and maximum sensor values is not enough to effectively calculate a good threshold.

• **Solution:** What is needed is a whole history of past sensor values, allowing the calculation of (for instance) the *average* sensor reading.

• Driver code available: install an interrupt routine that periodically samples the sensor values and stores them in a buffer. Other functions, such as the current maximum or current average functions, iterate through the stored values to calculate their results.