

COMP 551

ADVANCED ROBOTICS LAB

Lecture 06: Communications

James McLurkin
Rice University
jmclurkin@rice.edu

Binary

Representation

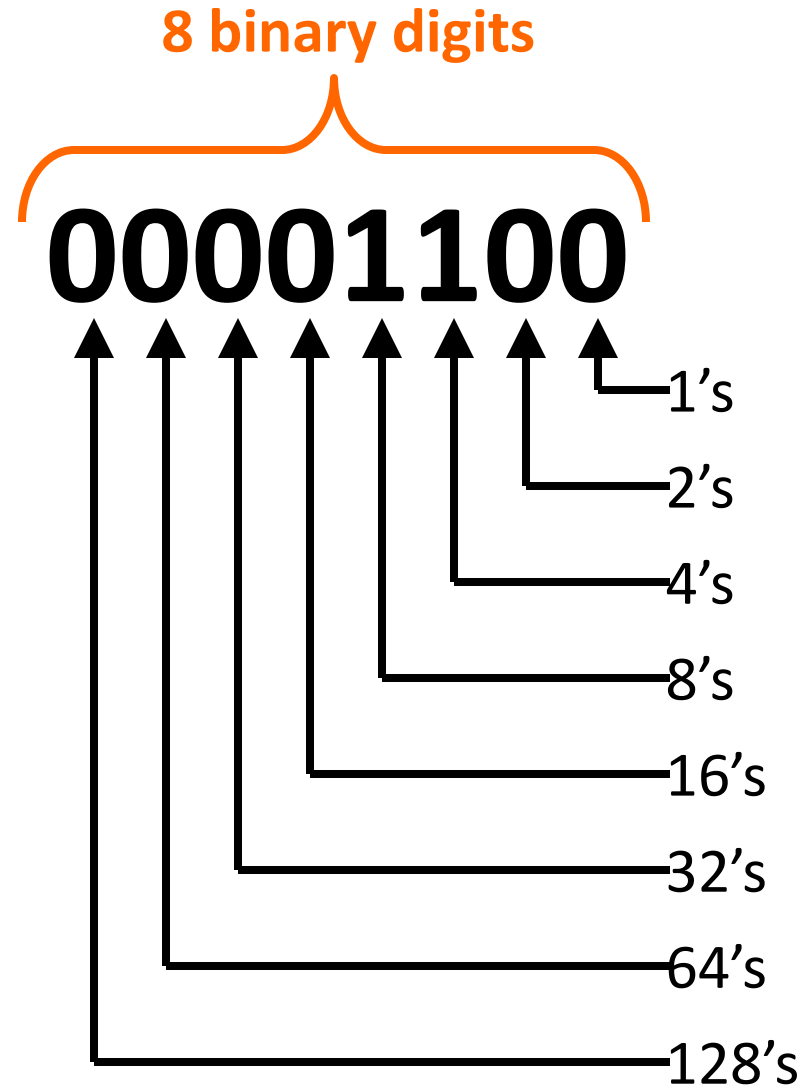
The Lowly Bit

10

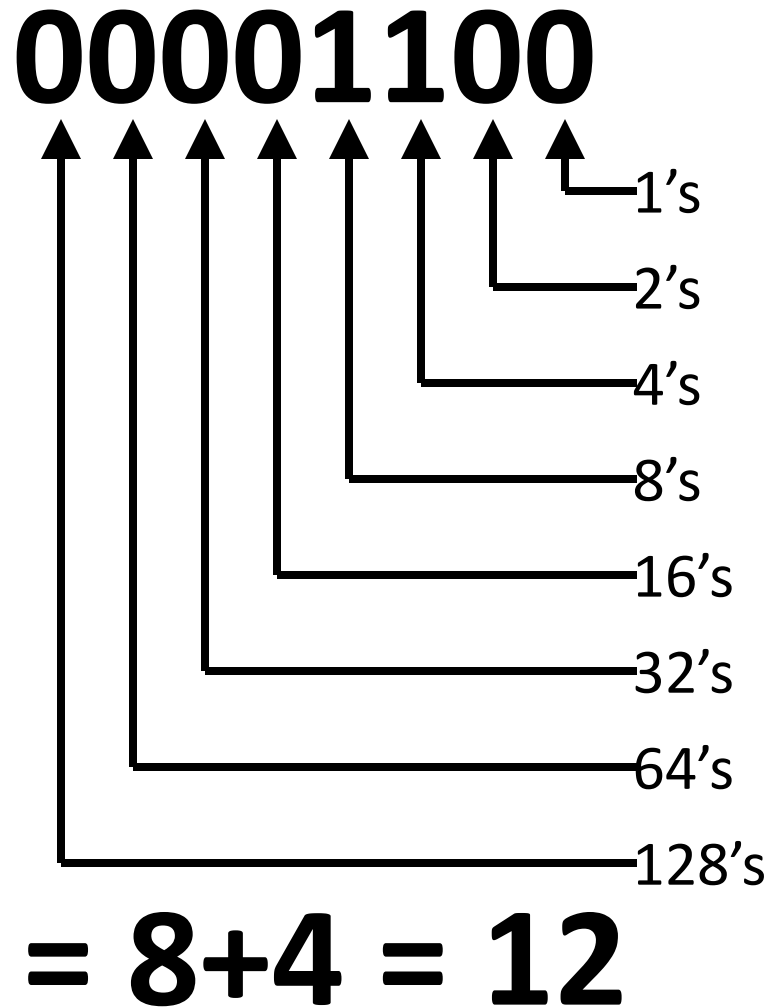
But they travel in swarms...

10

The Byte

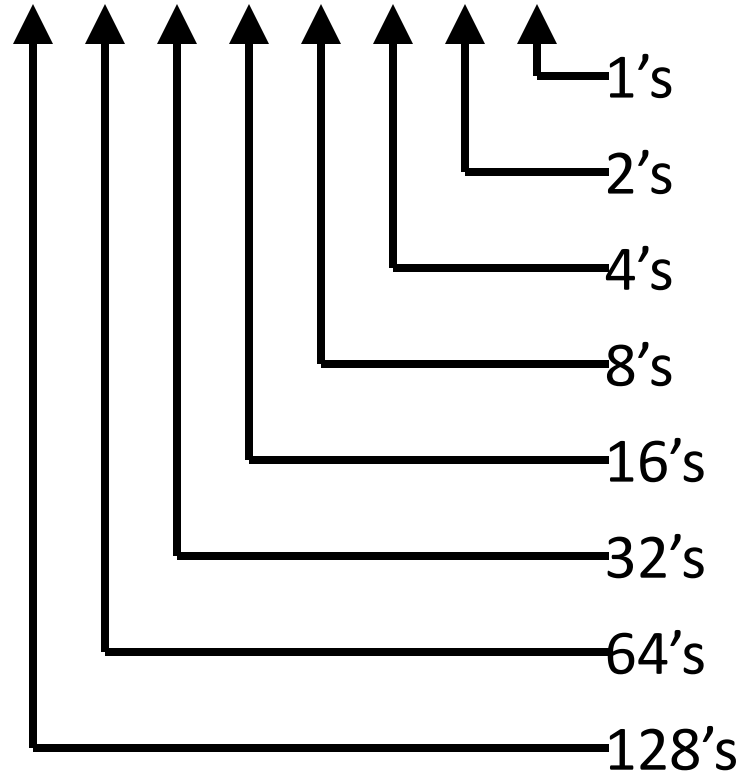


How can you read this quickly?



How can you read this quickly?

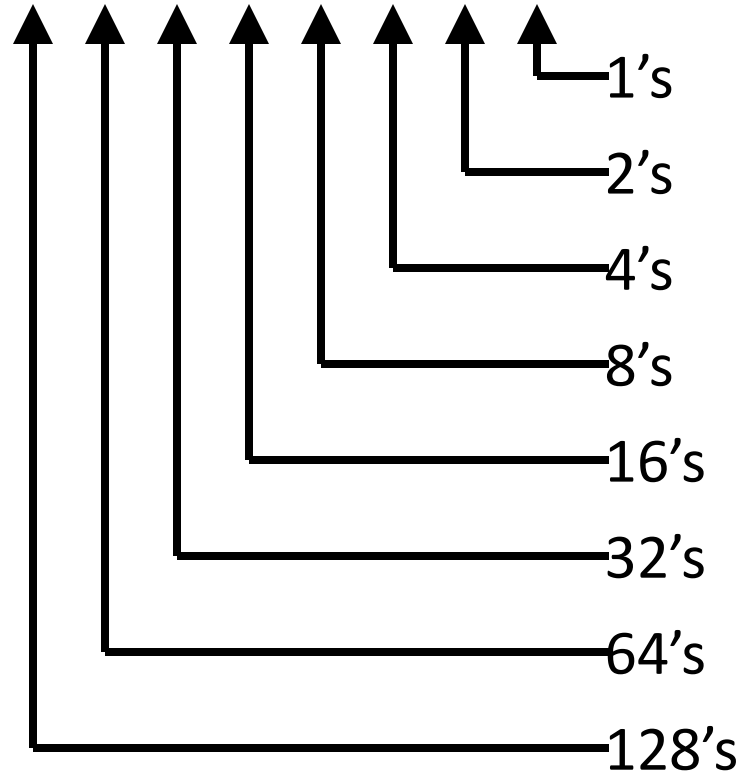
00100101



$$= 32 + 4 + 1 = 37$$

How can you read this quickly?

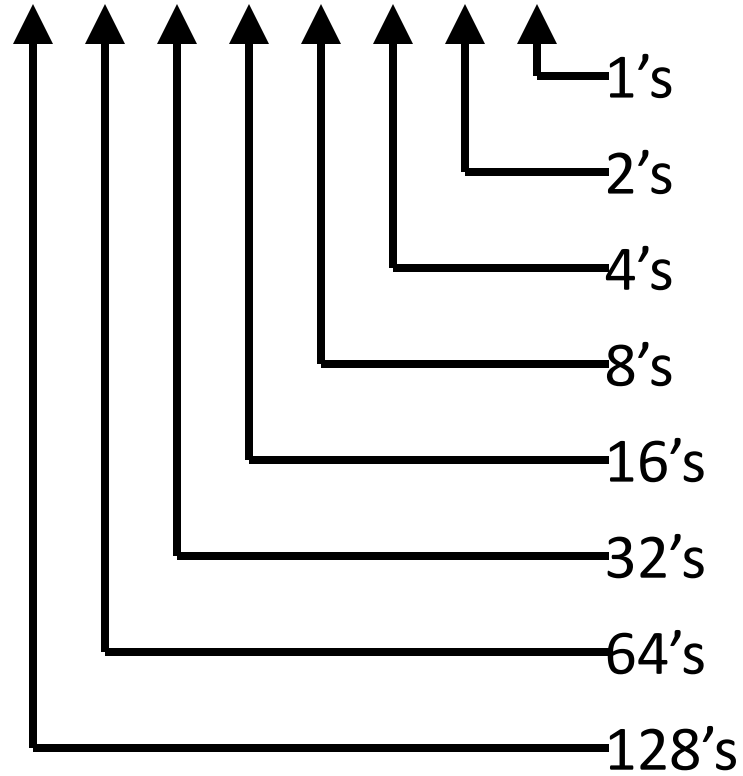
10000000



= 128 = 128

The Biggest, Baddest Byte:

11111111=255



Ok, so you can count, but can you add?

27 = 00011011

+26 = 00011010

ASCII Table

Easy. Assign a number to each letter.

| ASCII value | Character | ASCII value | Character | ASCII value | Character |
|-------------|-----------|-------------|-----------|-------------|-----------|
| 032 | (space) | 064 | @ | 096 | . |
| 033 | ! | 065 | A | 097 | a |
| 034 | " | 066 | B | 098 | b |
| 035 | # | 067 | C | 099 | c |
| 036 | \$ | 068 | D | 100 | d |
| 037 | % | 069 | E | 101 | e |
| 038 | & | 070 | F | 102 | f |
| 039 | ' | 071 | G | 103 | g |
| 040 | (| 072 | H | 104 | h |
| 041 |) | 073 | I | 105 | i |
| 042 | * | 074 | J | 106 | j |
| 043 | + | 075 | K | 107 | k |
| 044 | , | 076 | L | 108 | l |
| 045 | - | 077 | M | 109 | m |
| 046 | . | 078 | N | 110 | n |
| 047 | / | 079 | O | 111 | o |
| 048 | 0 | 080 | P | 112 | p |
| 049 | 1 | 081 | Q | 113 | q |
| 050 | 2 | 082 | R | 114 | r |
| 051 | 3 | 083 | S | 115 | s |
| 052 | 4 | 084 | T | 116 | t |
| 053 | 5 | 085 | U | 117 | u |
| 054 | 6 | 086 | V | 118 | v |
| 055 | 7 | 087 | W | 119 | w |
| 056 | 8 | 088 | X | 120 | x |
| 057 | 9 | 089 | Y | 121 | y |
| 058 | : | 090 | Z | 122 | z |
| 059 | ; | 091 | [| 123 | { |
| 060 | < | 092 | \ | 124 | |
| 061 | = | 093 |] | 125 | } |
| 062 | > | 094 | ^ | 126 | ~ |
| 063 | ? | 095 | _ | 127 | ☐ |

Let's write some email!

A short email: Only use 4 letters

| ASCII value | Character | Control character | ASCII value | Character | ASCII value | Character | ASCII value | Character |
|-------------|-------------------|-------------------|-------------|-----------|-------------|-----------|-------------|-----------|
| 000 | (null) | NUL | 032 | (space) | 064 | @ | 096 | |
| 001 | ☺ | SOH | 033 | ! | 065 | A | 097 | a |
| 002 | ☻ | STX | 034 | " | 066 | B | 098 | b |
| 003 | ♥ | ETX | 035 | # | 067 | C | 099 | c |
| 004 | ♦ | EOT | 036 | \$ | 068 | D | 100 | d |
| 005 | ♣ | ENQ | 037 | % | 069 | E | 101 | e |
| 006 | ♠ | ACK | 038 | & | 070 | F | 102 | f |
| 007 | (beep) | BEL | 039 | ' | 071 | G | 103 | g |
| 008 | ▣ | BS | 040 | (| 072 | H | 104 | h |
| 009 | (tab) | HT | 041 |) | 073 | I | 105 | i |
| 010 | (line feed) | LF | 042 | * | 074 | J | 106 | j |
| 011 | (home) | VT | 043 | + | 075 | K | 107 | k |
| 012 | (form feed) | FF | 044 | , | 076 | L | 108 | l |
| 013 | (carriage return) | CR | 045 | - | 077 | M | 109 | m |
| 014 | ♪ | SO | 046 | . | 078 | N | 110 | n |
| 015 | ☼ | SI | 047 | / | 079 | O | 111 | o |
| 016 | ▾ | DLE | 048 | 0 | 080 | P | 112 | p |
| 017 | ▴ | DC1 | 049 | 1 | 081 | Q | 113 | q |
| 018 | ↕ | DC2 | 050 | 2 | 082 | R | 114 | r |
| 019 | !! | DC3 | 051 | 3 | 083 | S | 115 | s |
| 020 | π | DC4 | 052 | 4 | 084 | T | 116 | t |
| 021 | \$ | NAK | 053 | 5 | 085 | U | 117 | u |
| 022 | ▬ | SYN | 054 | 6 | 086 | V | 118 | v |
| 023 | ↕ | ETB | 055 | 7 | 087 | W | 119 | w |
| 024 | ↕ | CAN | 056 | 8 | 088 | X | 120 | x |
| 025 | ↕ | EM | 057 | 9 | 089 | Y | 121 | y |
| 026 | ↕ | SUB | 058 | : | 090 | Z | 122 | z |
| 027 | ← | ESC | 059 | : | 091 | [| 123 | { |
| 028 | (cursor right) | FS | 060 | < | 092 | \ | 124 | |
| 029 | (cursor left) | GS | 061 | = | 093 |] | 125 | } |
| 030 | (cursor up) | RS | 062 | > | 094 | ^ | 126 | ~ |
| 031 | (cursor down) | US | 063 | ? | 095 | - | 127 | ▯ |

Ok, The email's done

Now how do we send it from robot to robot?



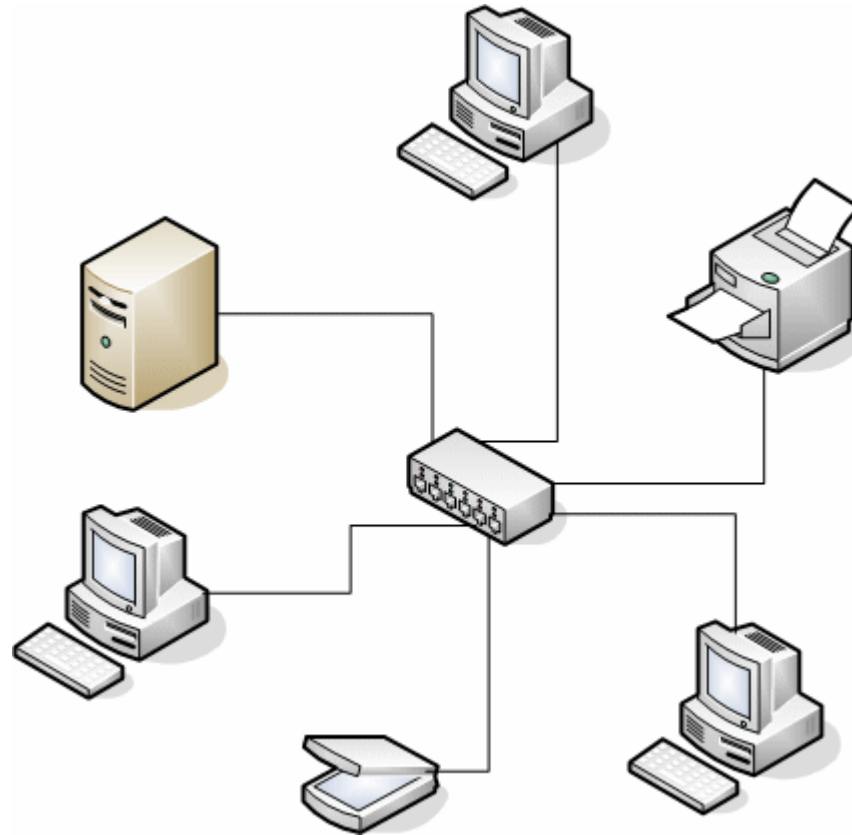
wazzup!



*That is
so 90's*

Digital Communications

We have digital communication Computers are networked



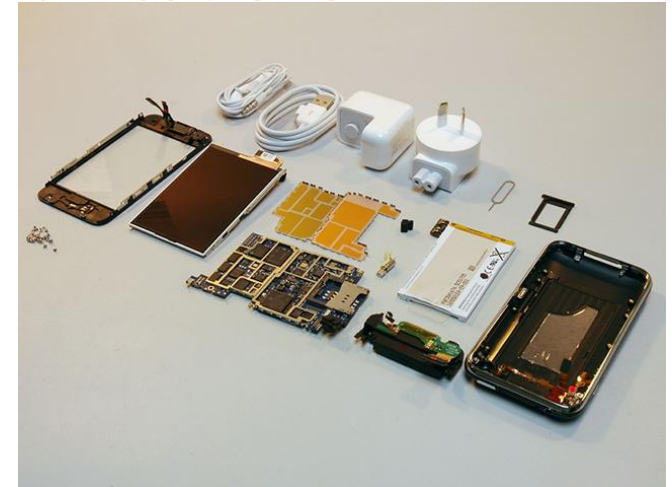
We're surrounded by Digital Communications



Ethernet



USB



3G and WiFi
(This is an iPhone)



WiFi



Remote Control

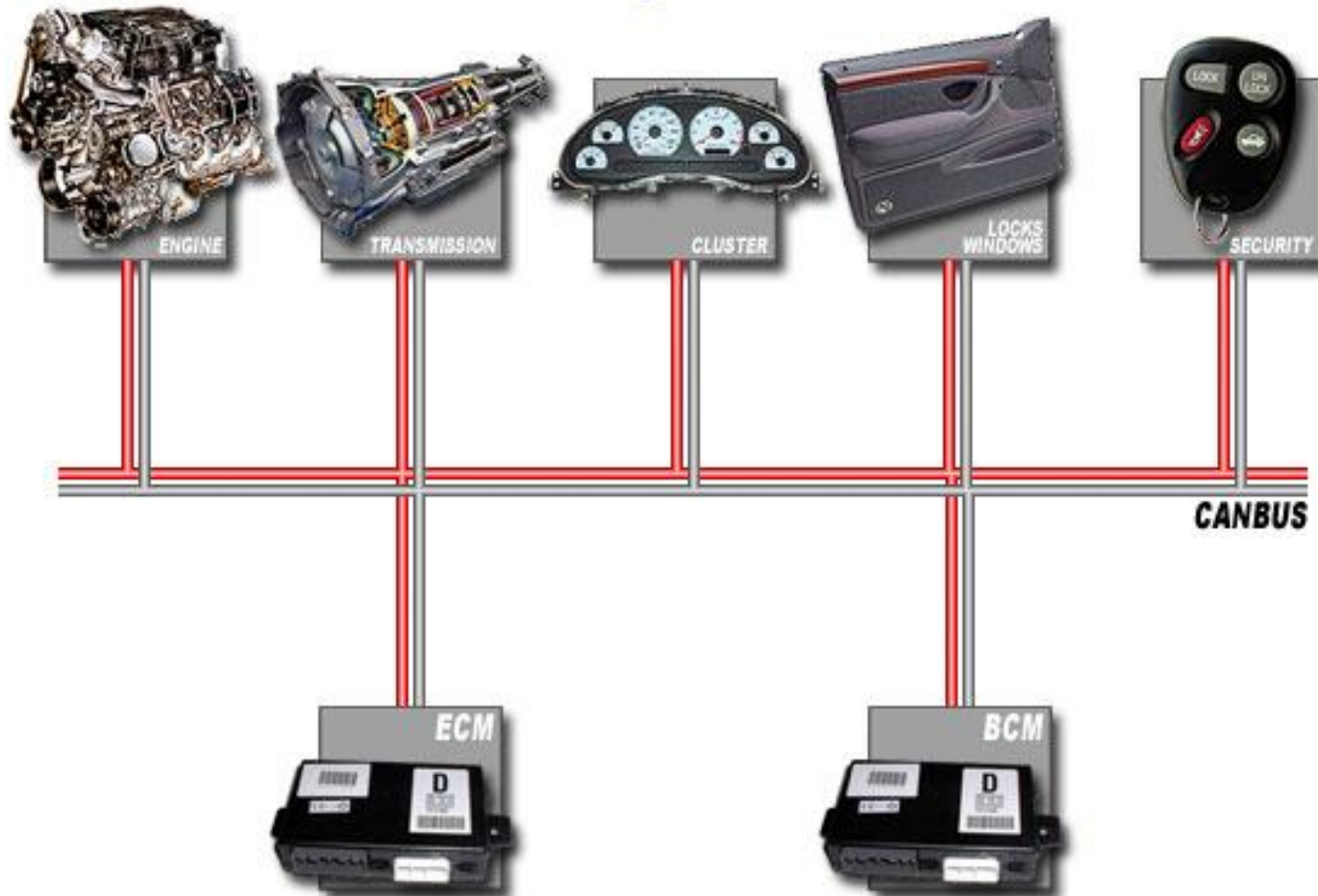


r-one robot

CAN Bus



Vehicle Wiring: CAN Bus network



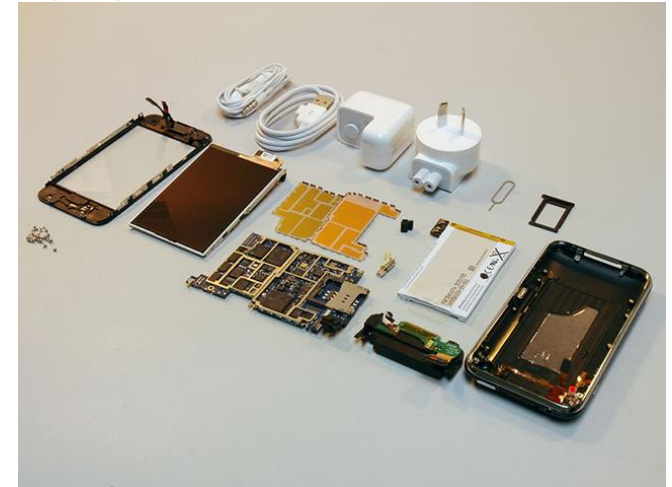
Physical Representation of Digital Data



Ethernet
Voltage



USB
Voltage



3G and WiFi
radio waves



WiFi
radio waves



Remote Control
IR Light



r-one robot
IR Light and radio

Bandwidth

This is a measure of the amount of bits we can transmit over a *channel*

- Its units are BPS, Bits Per Second

Anybody know some popular bandwidths?

- Ethernet = {10/100/1000} mbps
- USB 2.0 = 480 mbps
- Wi-Fi (802.11g) = 54 mbps
- r-one radio: 2 mbps, IR: 1.25 kbps

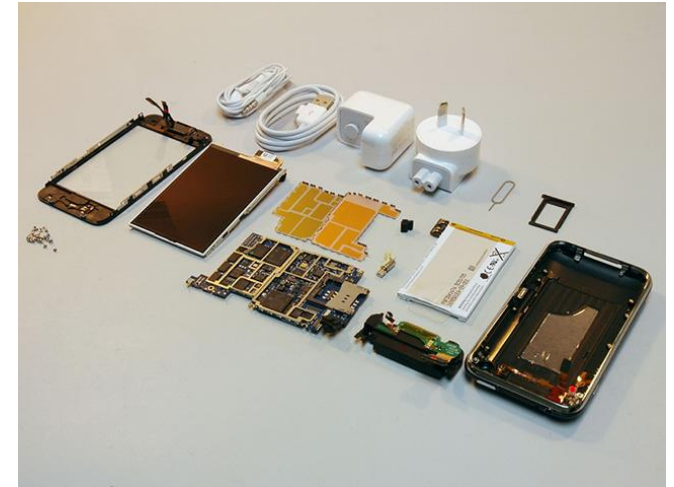
Bandwidth



Ethernet
1000 mbps



USB
480 mbps



3G and WiFi
384 kbps/54 mbps



WiFi
54 mbps



Remote Control
1.25 kbps



r-one robot
2 mbps/1.25 kbps

0110110111011001101010101010101011000110010010101011011011011
01011000110010010101011011011101100010101101010101010101100
01101101110110011010101010101010110001100100101010110110110110
010110001100100101010110110111011001010100110101010101010110
01101101110110011010101010101011000110010010101011011011011011
01011000110010010101011011011101100110101010101010110001100
01110110011010101010101011000110010010101011011011011011001101
000110010010101011011011101100110101010101010110001100100101
110011010101010101011000110010010101011011011001101010101
010010101011011011101100110101101010101011000110010010101011
01010101010101100011001001010110110110110011001101010101010
0101011011011011011011011011011011011011011011011011011011011
01010101100011001001010101101101101100110101010101010110001
01101110110011010101010101011000110010010101011011011101100
01100011001001010101101101101100110101010101010110001100100
1101100110101010101011000110010010101011011011101100110101
0110010010101011011011011001101010101010110001100100101010
0110101010101011000110010010101011011011101100110101010101
001010101101101101100110101010101010110001100100101010110110
01010101011000110010010101011011011101100110101010101010110
0110110110110011010101010101011000110010010101011011011101

That's a lot of bits!

If we're sending so many bits so quickly, how do we tell them apart?

Telling one bit from another

A continuous stream of bits is hard to deal with.

Imagine a computer network: You might have many questions about these bits:

- Are these bits for you?
- Where did these bits come from?
- What do these bits mean?
- Are these bits error-free?

The Packet

A packet is a chunk of data with a well-defined beginning, end, and structure

A packet has four parts:

- Some kind of **start indication** that tells the network that a packet is starting
- Some kind of **header** that tells the network what the packet is, where it is from, and where it is going
- Some kind of **data**. That's kind of the point of the packet...
- Some kind of **error detection** to check the validity of the packet.



Start Indication: The Preamble:

Some easy-to-detect sequence of bits to alert the receiver that a packet is starting:

Start Indication: The Preamble:

Some easy-to-detect sequence of bits to alert the receiver that a packet is starting:

01010101

The Header: Routing Information

This tells the packet:

- what it is,
- where it is from, and
- where it is going

This is how everything knows where it's going

- On the internet,
- in your car,
- inside your robots,
- in between your robots via infra-red (IR),
- and in between the robots and your computer via USB

Detecting Errors

There are many sources of error in communications networks

- Errors can make our message invalid
- We highlight two: Noise and Collisions

Noise:

- Somebody comes along and changes your signal.
- They can change voltage, add radio waves, or mess with your light

Collisions:

- Somebody comes along and tries to talk at the same time
- Both messages are lost

Error Detection

Say I send you a (very) short email with your grade in ENGI 128:

'A'

(Nice job!)

In ASCII, 'A' is 65, which is 01000001 in binary

But what if an error happens during communication, and a bit is flipped? You receive:

01000011

What does this mean?

How can we prevent the bit from being flipped?

What else can we do?

Error Detection

Let's send two pieces of information: The information, and something to let us know if the data is intact

The sender composes the message data, then computes another piece of information that summarizes the message. The sender transmits:

$$\text{check}_{\text{sender}} = f(\text{data})$$

$$\text{msg} = (\text{data}, \text{check}_{\text{sender}})$$

When the receiver gets this information, it computes its own copy of the check independently from the message data:

$$\text{check}_{\text{receiver}} = f(\text{data})$$

If $\text{check}_{\text{receiver}} == \text{check}_{\text{sender}}$, then all is well. If not, then we discard the entire message

What can we use for $f(\text{data})$?

Error Detection

Let's send two pieces of information: The information, and something to let us know if the data is intact

Plan A: We send the data again – the odds of getting two bits flipped in the same place are slim:

msg = 'A', which is 01000001 in binary

we send: **0100000101000001**

For a longer message, we need lots of extra bits to detect an error:

10010101 01100101 01011001 01010110 01010101 10010101 10010101 01100101 01011001 01010110 01010101

10010101 01100101 01011001 01010110 01010101 10010101 10010101 01100101 01011001 01010110 01010101

This will take twice as much bandwidth, and bandwidth is expensive

Can we do better?

Population Count

Plan B: How about we count the total number of one bits in the message?

msg = 'A', which is 01000001 in binary

we send: **0100000100000010**

This is better. For a longer message, we still only need 8 extra bits to detect an error:

10010101 01100101 01011001 01010110 01010101 10010101 10010101 01100101 01011001 01010110 01010101 **10010101**

And only use one byte for Checksum. Sweet!

But there is a problem...

Checksums Behaving Badly

Plan B: How about we count the total number of bits in the message?

msg = 'A', which is 01000001 in binary

we send: **0100000100000010**

we receive: **0100001000000010**

What is the problem?

Cyclic Redundancy Check (CRC)

Plan C: How about we make a fancy polynomial that is optimized to detect the most common errors on our communication channel?

Duh... Of course this is what we should do!

msg = 'A', which is 01000001 in binary

they send: **01000001** **01101010**

we receive: **01000011** **01101010**

we compute our CRC: **11101011**

They don't match. Ta-da: Error detection!

Start Indication: The Preamble:

Some easy-to-detect sequence of bits to alert the receiver that a packet is starting:

01010101

Start Indication: The Preamble, revisited

Some easy-to-detect sequence of bits to alert the receiver that a packet is starting:

01010101

01010101

01010101

01010101 **101101110110**

01010101

01010101

01010101

r-one IR
Communications

The Packet

A packet is a chunk of data with a well-defined beginning, end, and structure

A packet has four parts:

- Some kind of **start indication** that tells the network that a packet is starting
- Some kind of **header** that tells the network what the packet is, where it is from, and where it is going
- Some kind of **data**. That's kind of the point of the packet...
- Some kind of **error detection** to check the validity of the packet.



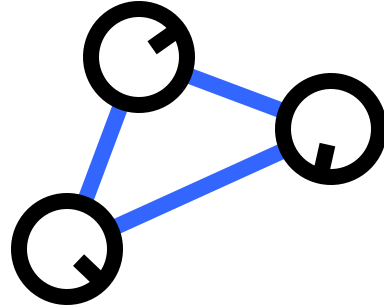
The nitty-gritty

How do these bits actually get from robot to robot?



Inter-Robot Communications

When the robots are moving, how often should they communicate?

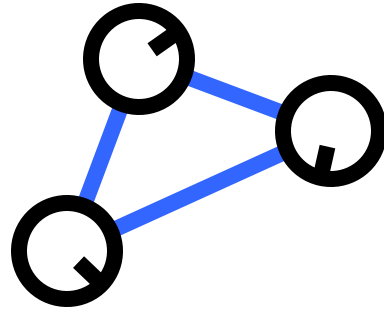


- a. all the time
- b. when they get to their goal positions
- c. only when they need to

Periodic Communications

Ok, so they need to communicate all the time

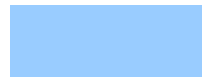
But how frequently?

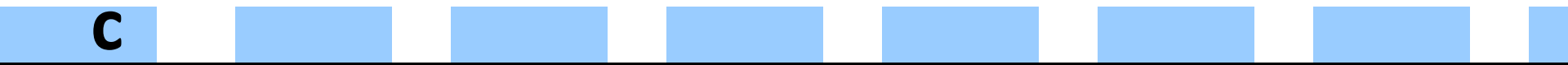
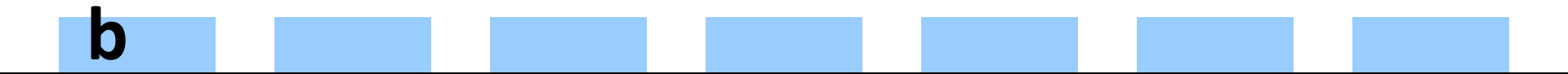
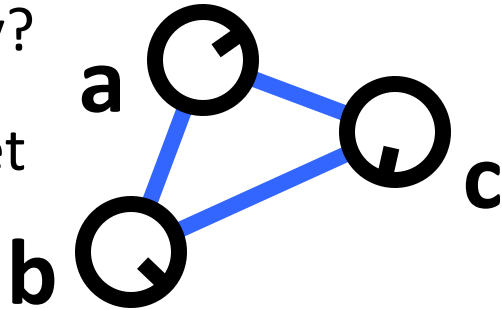


Periodic Communications

Ok, so they need to communicate all the time

But how frequently?

 = one packet

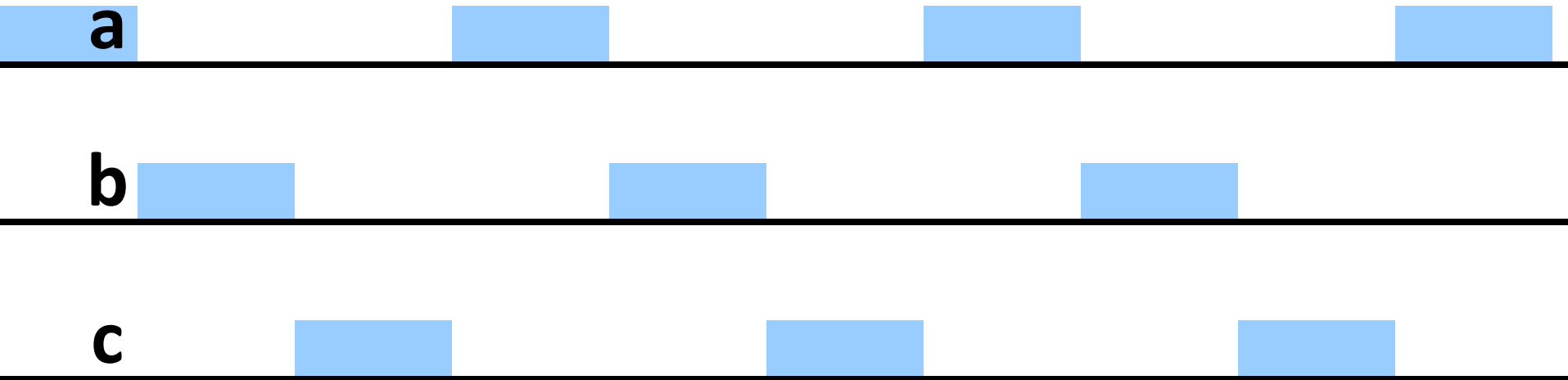
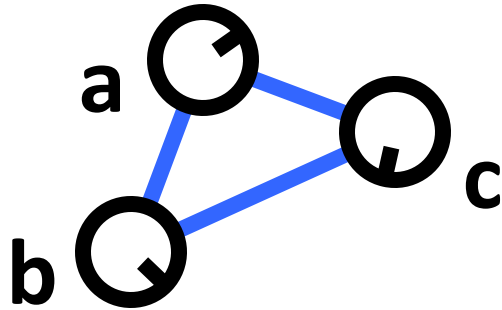


t \longrightarrow

Periodic Communications

We can avoid collisions with proper spacing

But now what is our problem?

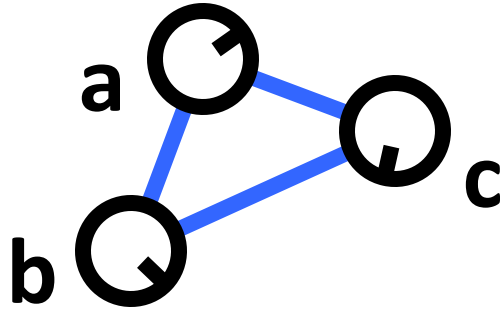


Periodic Communications

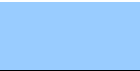
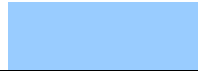
Ok, we'll be more relaxed about the timing.

But we waste $\frac{1}{2}$ of the bandwidth!

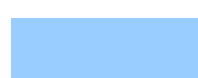
Is this worth the trade-off?



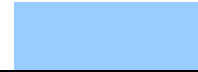
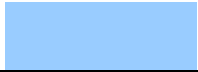
a



b



c



The Neighbor “Round”

Each robot broadcasts its information to its neighbors at periodic intervals

- This period is fixed across all robots, but the starting time is random
- This gives probabilistic assurances (not guarantees) that messages won't collide (Abramson, Aloha protocol, 1970(!))

If messages are 23ms and the round is 230ms:

1. How many neighbors can each robot have?
2. What if the messages collide?
3. What if they all start at exactly the same time?
4. How does ethernet handle collisions?
5. Why can't we use this technique?

Measuring Local Network Geometry

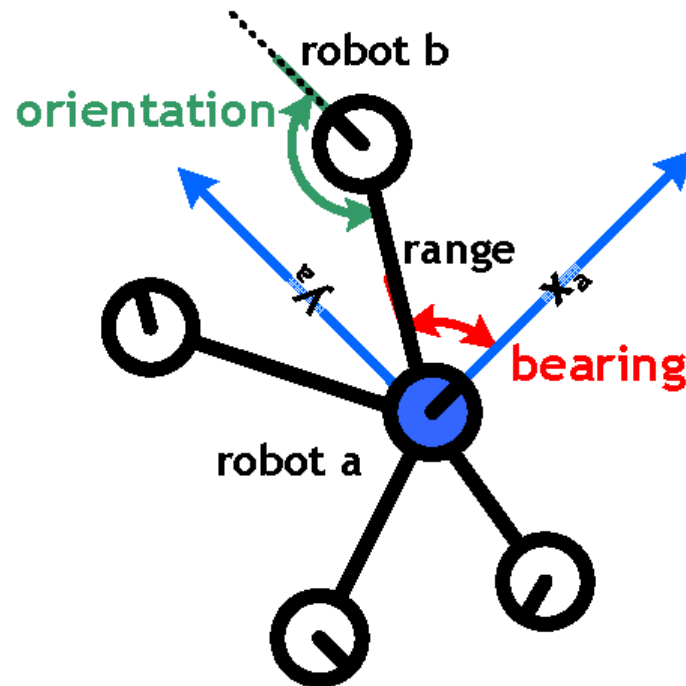
Local Coordinates

Measure *pose* of robot b in robot a's coordinate system

$$\text{pose} = (x, y, \theta)$$

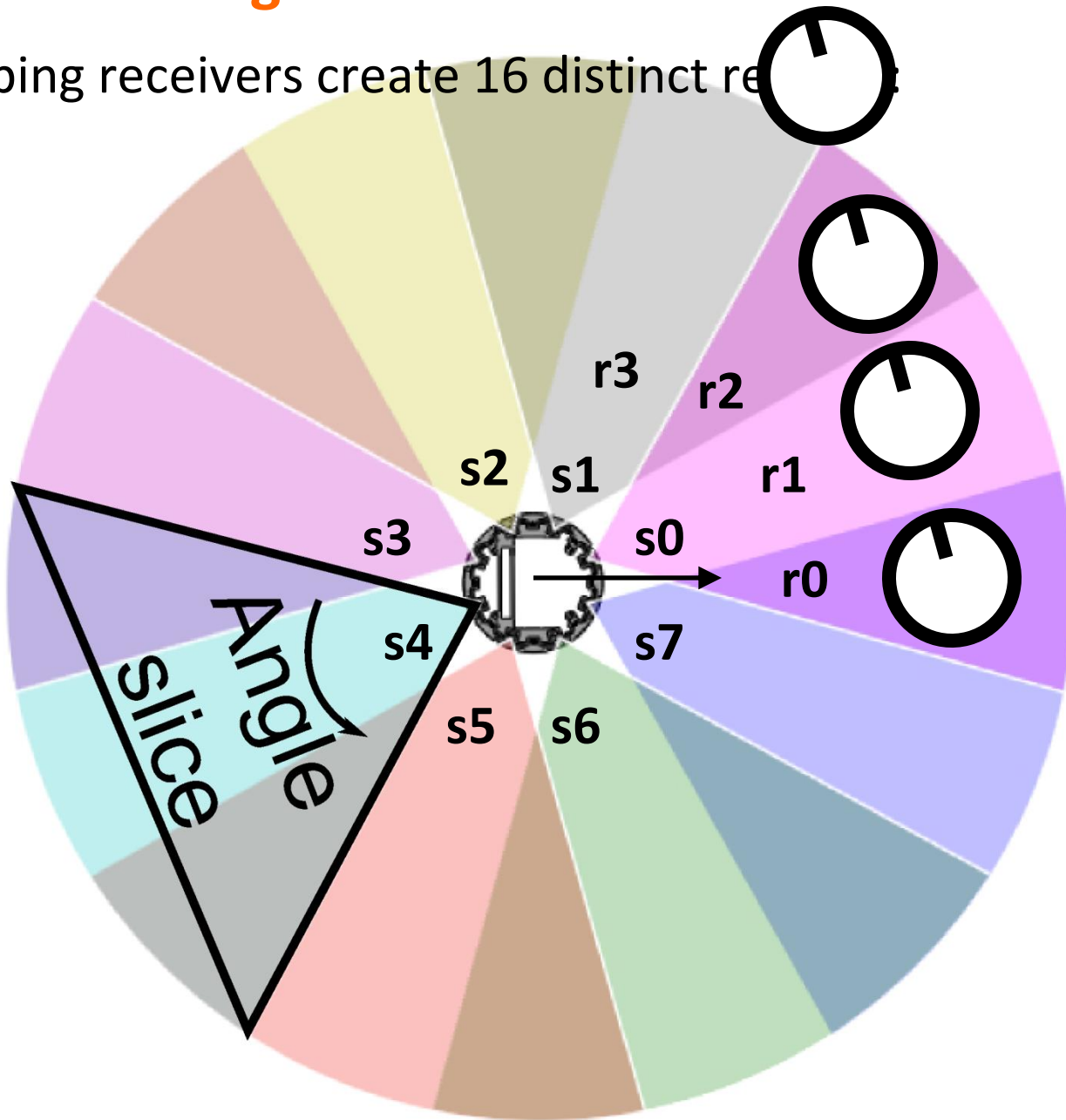
-or-

$$\text{pose} = (\text{range}, \text{bearing}, \text{orientation})$$



IR Sectors: Bearing Measurement

8 overlapping receivers create 16 distinct receivers:

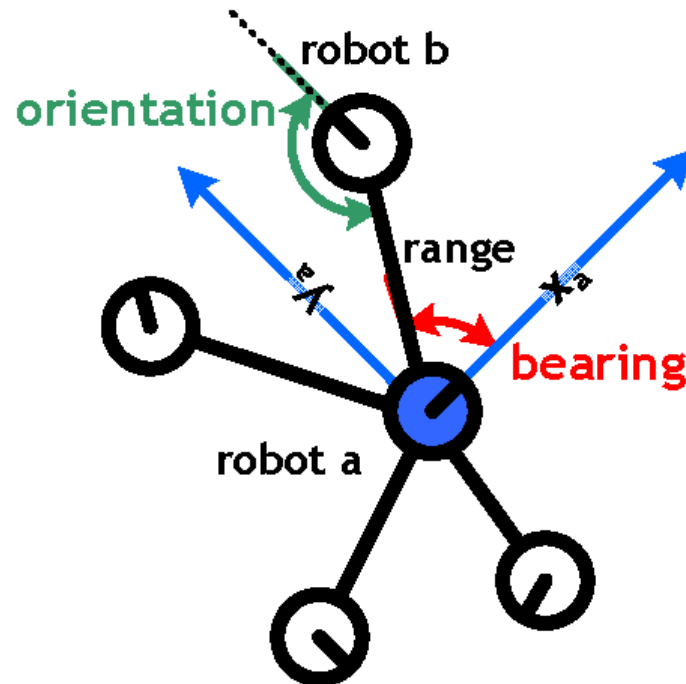


Local Coordinates: Bearing

We can measure bearing directly

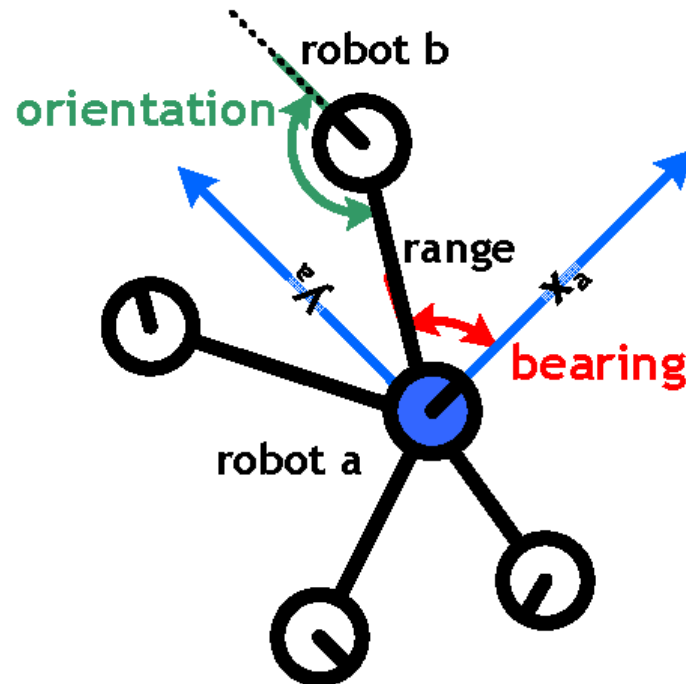
- With an accuracy of around $\pi/8$

What else can we measure?



Local Coordinates: Orientation

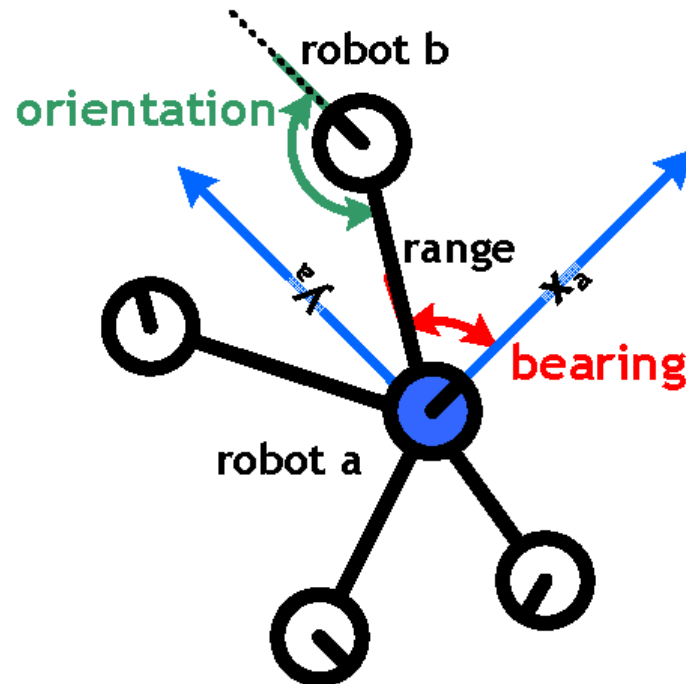
How can we measure the orientation of robot b from robot a's point of view?



Local Coordinates: Orientation

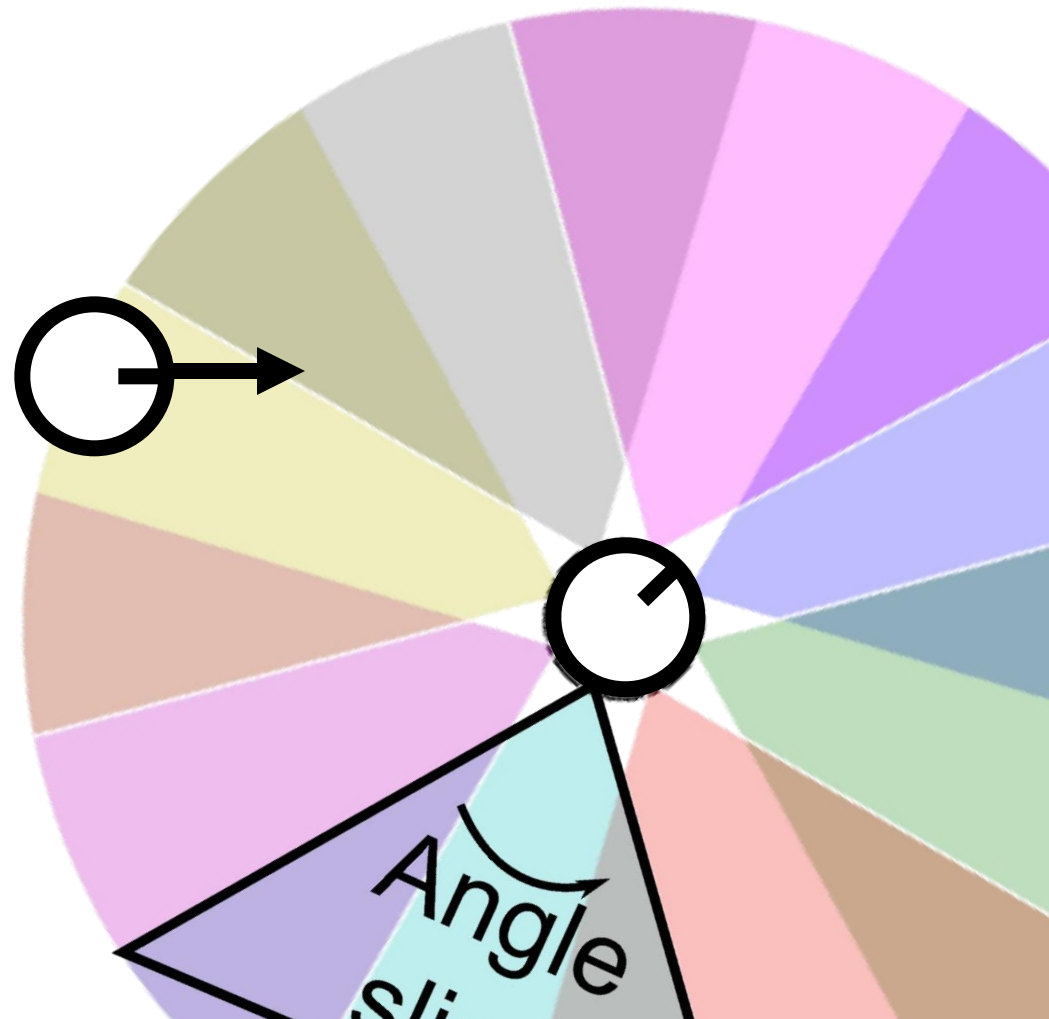
Note that the orientation of robot b from robot a's point of view is the same as the bearing of robot a from robot b's point of view

How can we get this information from robot a to robot b?



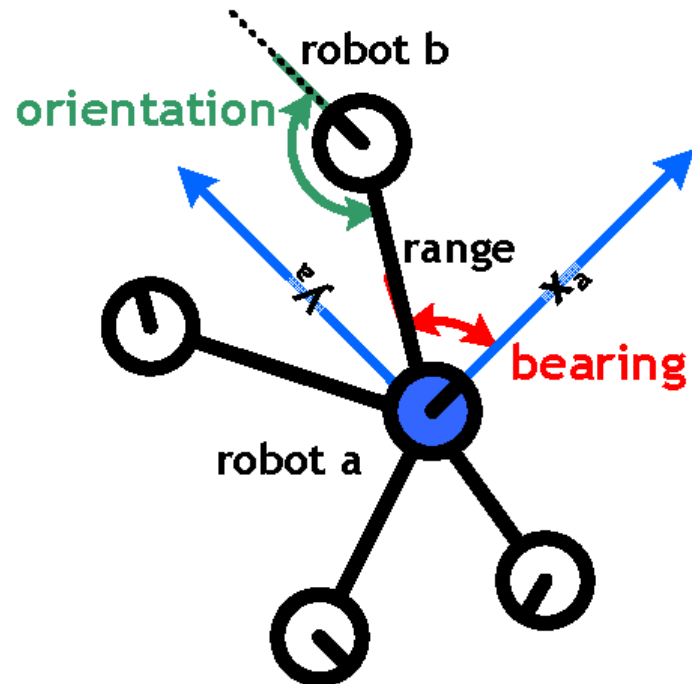
IR Sectors: Orientation Measurement

8 overlapping *transmitters* also create 16 distinct regions:



Local Coordinates: Distance

So what about distance?



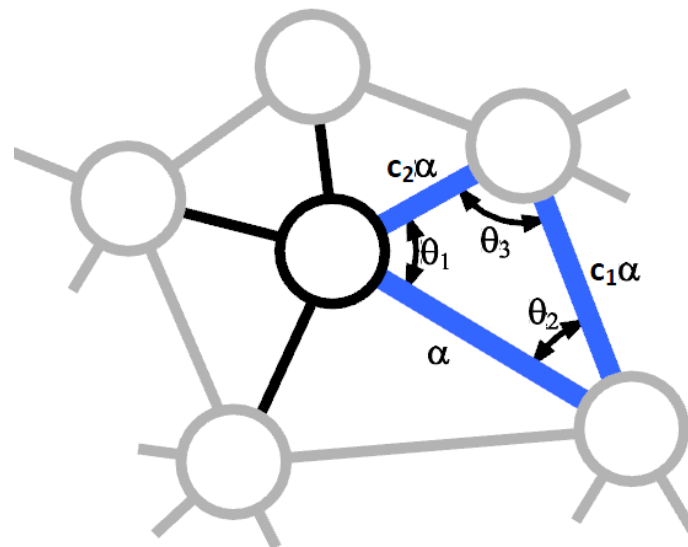
Scale-Free Coordinates

Can use *angular graph rigidity* to compute poses of neighbors

- Find triangles in the network
- Find the angles around this triangle
- This triangle is *rigid*: it can change size, but not shape

It's not perfect

- Can only compute pose up to an unknown scaling constant, α



a rigid 3-cycle

Particle Filters

We can produce an estimate of the range over time

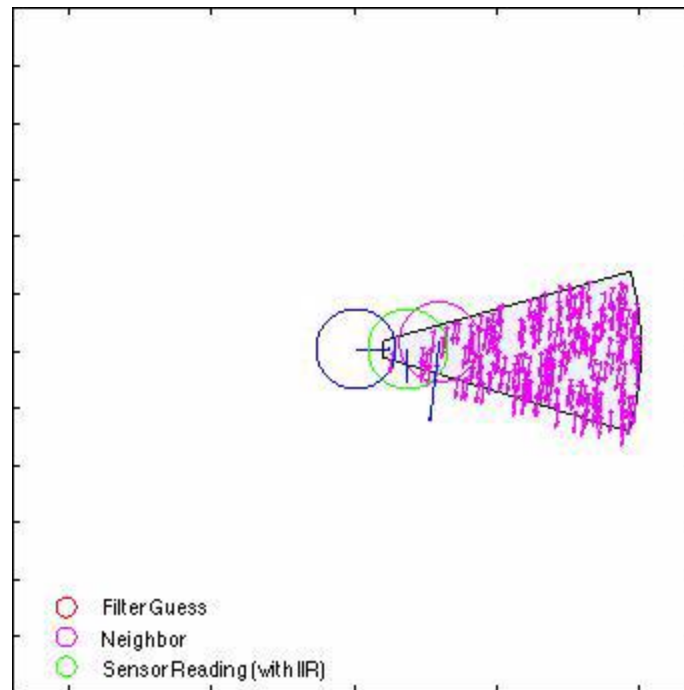


Local Coordinates: Improving Pose Estimates

Different sensors behave differently

- Our IR sensor resolution is limited.
- But our encoders are very precise

We can keep track of our neighbors motion to estimate where the robot is between sensor updates:



Bearing Motion Controller

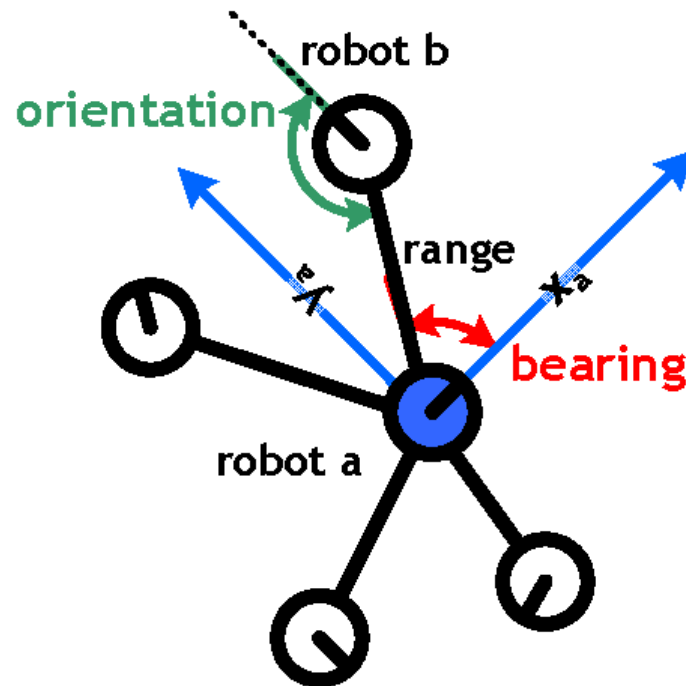
Local Coordinates

Measure *pose* of robot b in robot a's coordinate system

$$\text{pose} = (x, y, \theta)$$

-or-

$$\text{pose} = (\text{range}, \text{bearing}, \text{orientation})$$

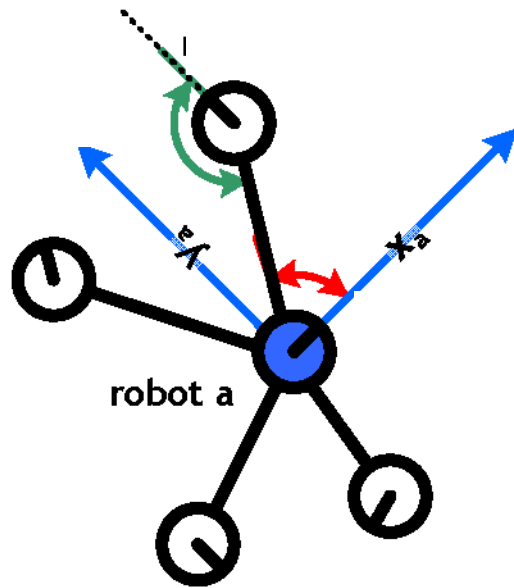


Motion Control

Instead of controlling left and right motors, it's often more convenient to control

- Translational velocity: t_v , and
- Rotational velocity, r_v

How can we compute left and right velocities from t_v and r_v ?



$$v_{\text{left}} = t_v - r_v$$

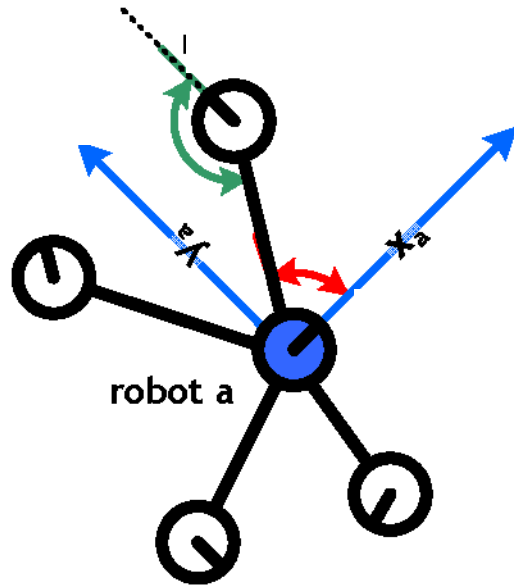
$$v_{\text{right}} = t_v + r_v$$

Motion Control

Instead of controlling left and right motors, it's often more convenient to control

- Translational velocity: v_t , and
- Rotational velocity, v_r

How can we compute left and right velocities from v_t and v_r ?

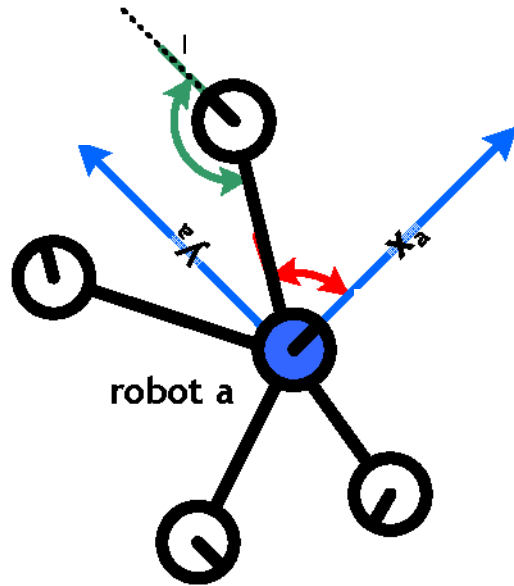


$$v_{\text{left}} = v_t - v_r * \text{WHEEL_BASE} / 2$$

$$v_{\text{right}} = v_t + v_r * \text{WHEEL_BASE} / 2$$

Motion Control

So, if we know the pose of a neighbor, can we compute r_v and t_v to get us there?



Multi-Hop Ad-hoc Networks

Communications is Necessary

This is the main difference between multi-robot systems and single robot systems

- All the data is not on a single computer
- Some kind of communications is required to cooperate

The communication cost of getting the data onto a single computer is often high

- Often too high to pay
- Multi-robot communications needs to consolidate this data somehow

OSI 7-Layer Network Stack

7. application



6. presentation



5. session



4. transport



3. network



2. data link



1. physical

Concepts

End-to-end Principle

- Put the intelligence at the ends
- Let the stuff in the middle be stupid

Best-Effort Routing

- “We’ll do our best to deliver your packet”
- No guarantee on time or path

Time

- Asynchronous: Ethernet CSMA, Exponential Random backoff
- Synchronous: Big global clock, distributed synchronizers

Routing

Flooding

- simple
- lots of wasted messages

Routing tables

- distance vector (RIP) routing
- each node has a routing table to all other nodes

Hierarchical routing

- requires much smaller routing tables
- need asymmetrical hardware to handle lots of packets
- but what if the network is changing frequently?

Graphical routing

- You can use the position of the node as a type of hierarchy
- Get packet close to the destination, then simple routing tables can do the rest

Ad-hoc networks

Sensor networks

- simple hardware
- limited bandwidth and processing

Building trees

- Use a broadcast flood
- Build efficient routing structures dynamically

r-one Summary

7. application: **Shared memory API**

6. presentation: n/a

5. session: n/a

4. transport: n/a

3. network: **Broadcast trees**

2. data link: **24-bit packets**

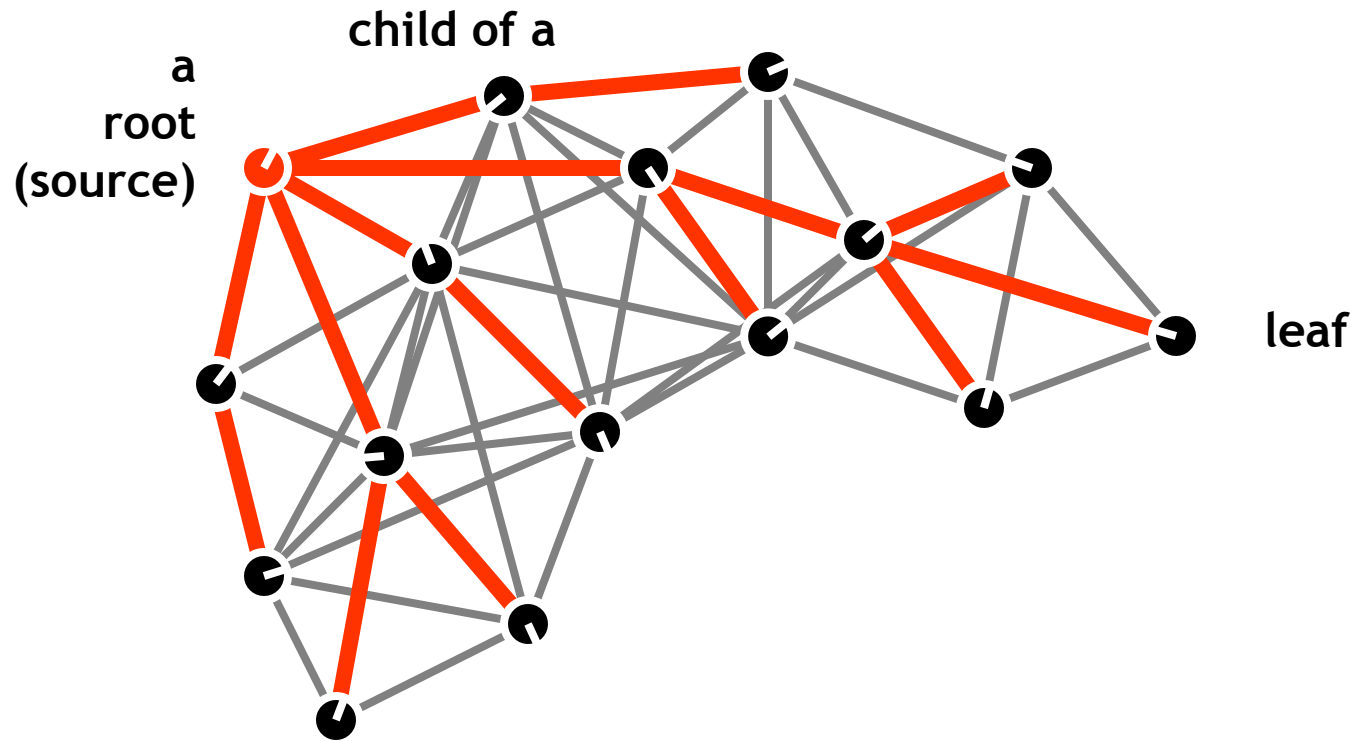
1. physical: **IR with Frequency encoding**

Communications in Multi-Robot Networks

Broadcast Tree Construction

Broadcast flood

Distributed BFS



$$G = \{V, E\}$$

$$|E| = |V| - 1$$

[white board]

Trees, root, leaves, parents

Basic theorems

Message Speed



$n = 44$, $\tau = 0.250s$, speed = 0, RSR = 0

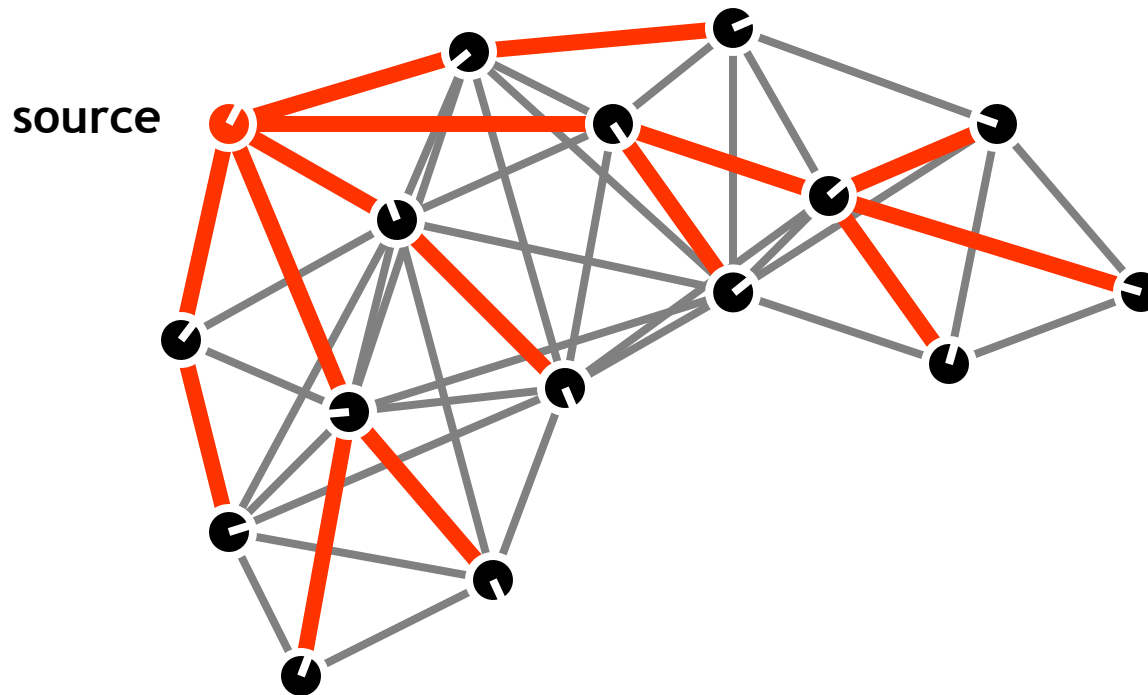
Broadcast Tree Path Distance

Purpose:

- To estimate the distance to the source of a message

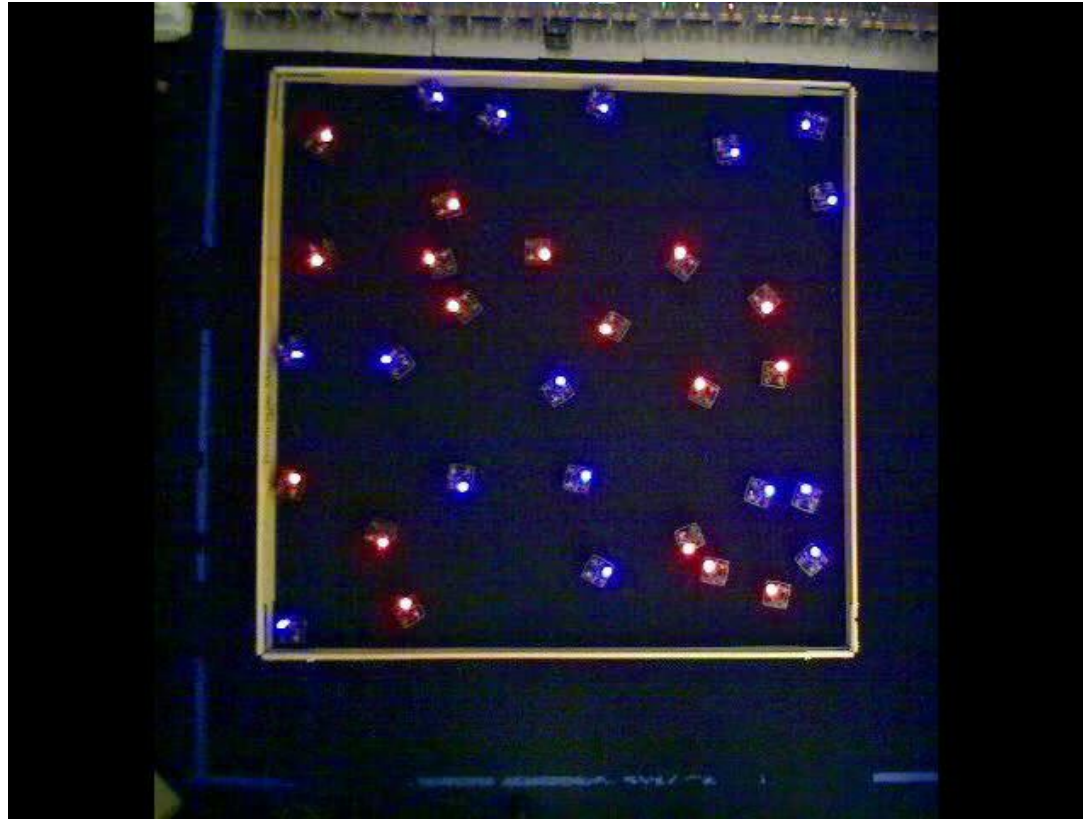
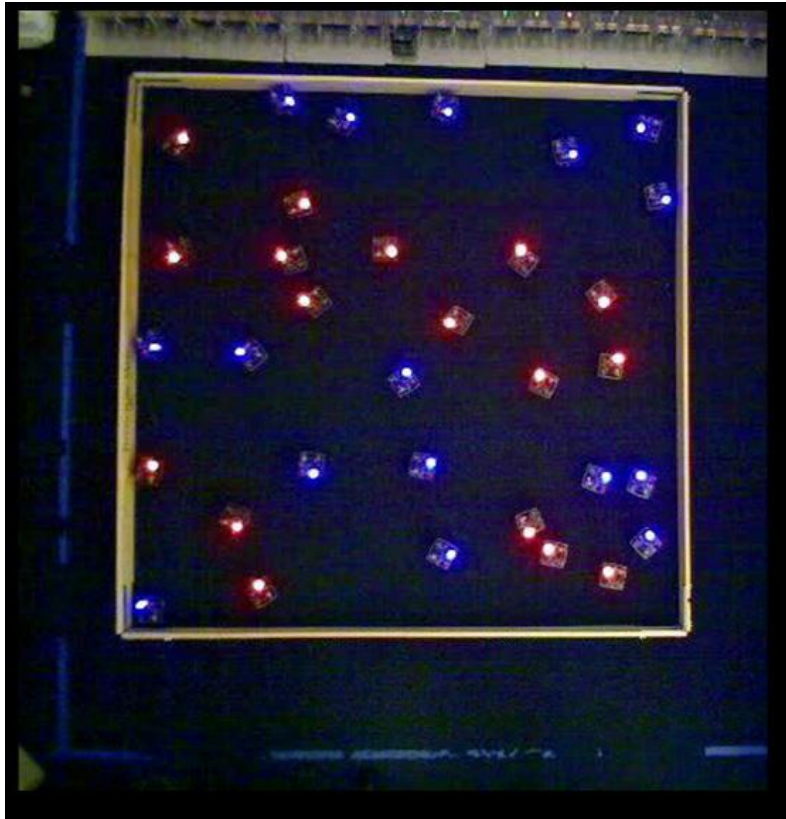
Accuracy Metric:

- correlation coefficient between measured and actual distances



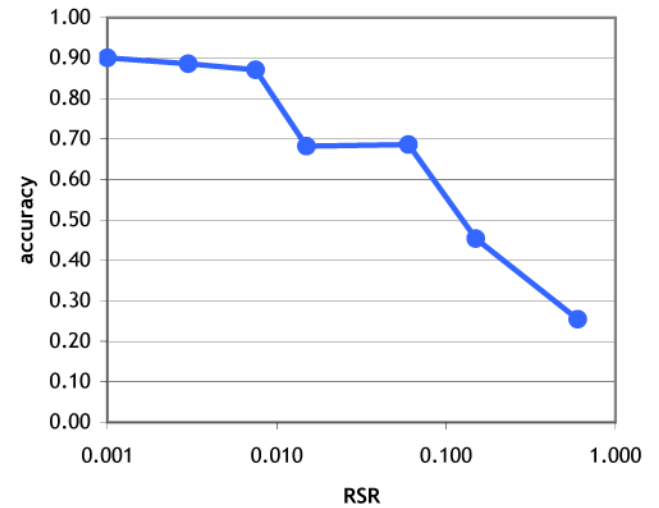
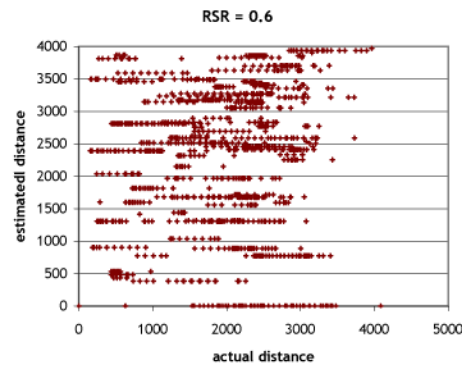
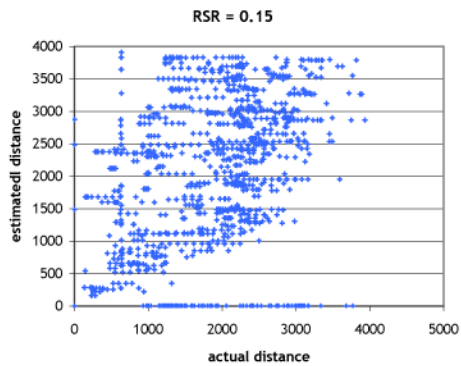
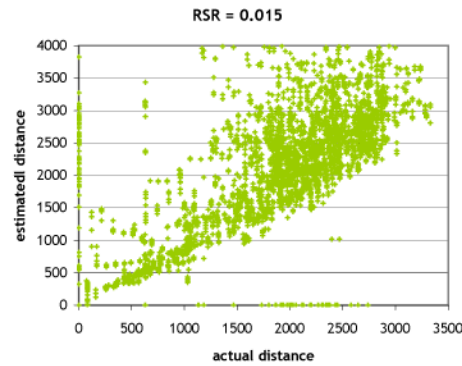
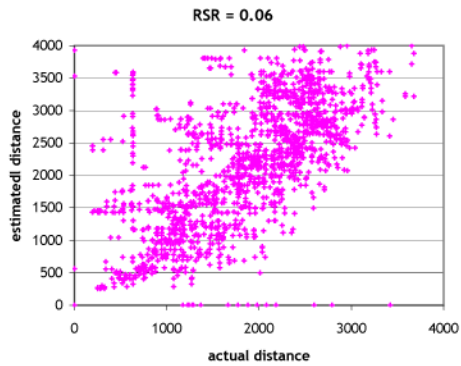
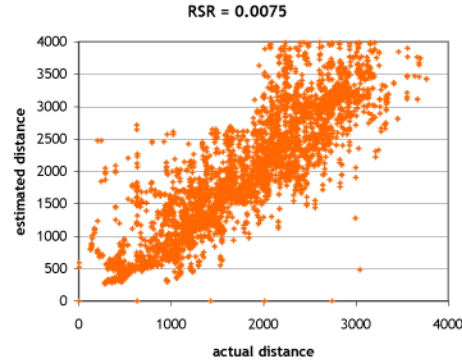
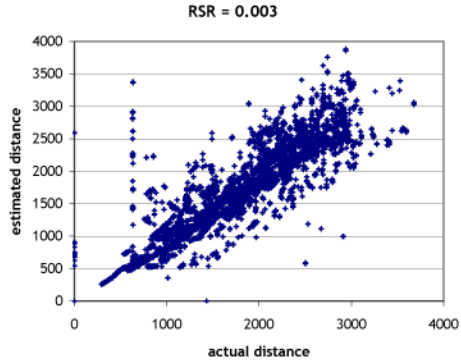
Dynamic Networks

Motion “smears” network and reduces the correspondence between network topology and physical positions



$n = 33$, $\tau = 0.250s$, speed = 0.08, RSR = 0.02

Broadcast Tree Path Distance



PS02: Follow the Leader and Flocking



Follow the leader