



# ENGI 128

INTRODUCTION TO ENGINEERING SYSTEMS

Lecture 14:  
r-one Communications Details,  
Measuring Network Geometry  
“Understand Your Technical World”

# Representing Digital Information

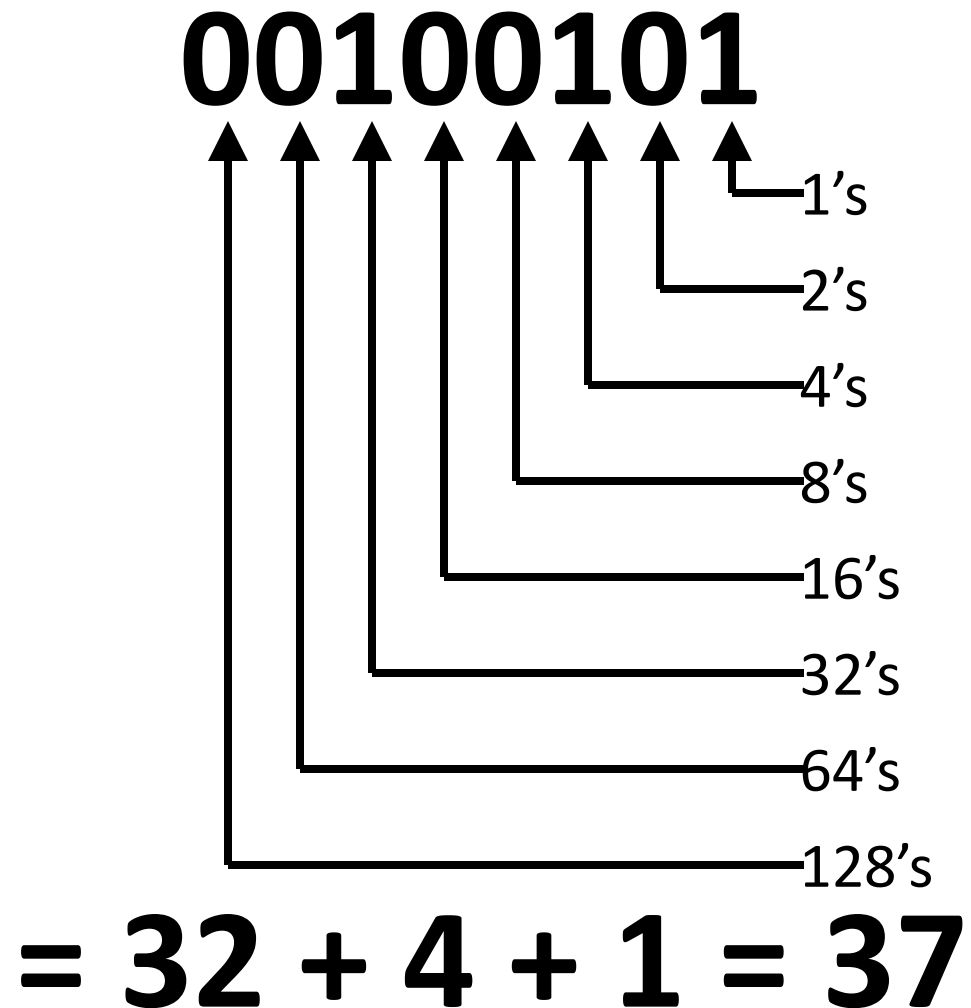
## The Lowly Bit

0

# The Lowly Bit

1

How can you read this quickly?



## Addition

$$\begin{array}{r} 00011011 \\ +00011010 \\ \hline 00110101 \end{array}$$

# Text: ASCII Table

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	

`@' = 64 = 01000000  
`A' = 65 = 01000001  
`B' = 66 = 01000010  
`C' = 67 = 01000011  
`C' = 67 = 01000100  
`C' = 67 = 01000101  
`C' = 67 = 01000110  
`C' = 67 = 01000111  
`@' = 64 = 01001000  
`A' = 65 = 01001001  
`B' = 66 = 01001010  
`C' = 67 = 01001011  
`C' = 67 = 01001100  
`C' = 67 = 01001101  
`C' = 67 = 01001110  
`C' = 67 = 01001111

...

# Text: ASCII Table (and counting)

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	

`@' = 64 = 01000000  
'A' = 65 = 01000001  
'B' = 66 = 01000010  
'C' = 67 = 01000011  
'D' = 68 = 01000100  
'E' = 69 = 01000101  
'F' = 70 = 01000110  
'G' = 71 = 01000111  
'H' = 72 = 01001000  
'I' = 73 = 01001001  
'J' = 74 = 01001010  
'K' = 75 = 01001011  
'L' = 76 = 01001100  
'M' = 77 = 01001101  
'N' = 78 = 01001110  
'O' = 79 = 01001111

...

# Bandwidth



Ethernet  
**1000 mbps**



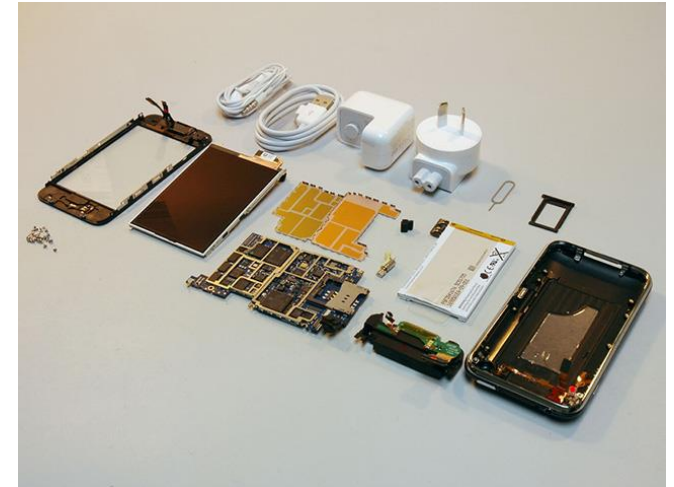
USB  
**480 mbps**



WiFi  
**54 mbps**



Remote Control  
**1.25 kbps**

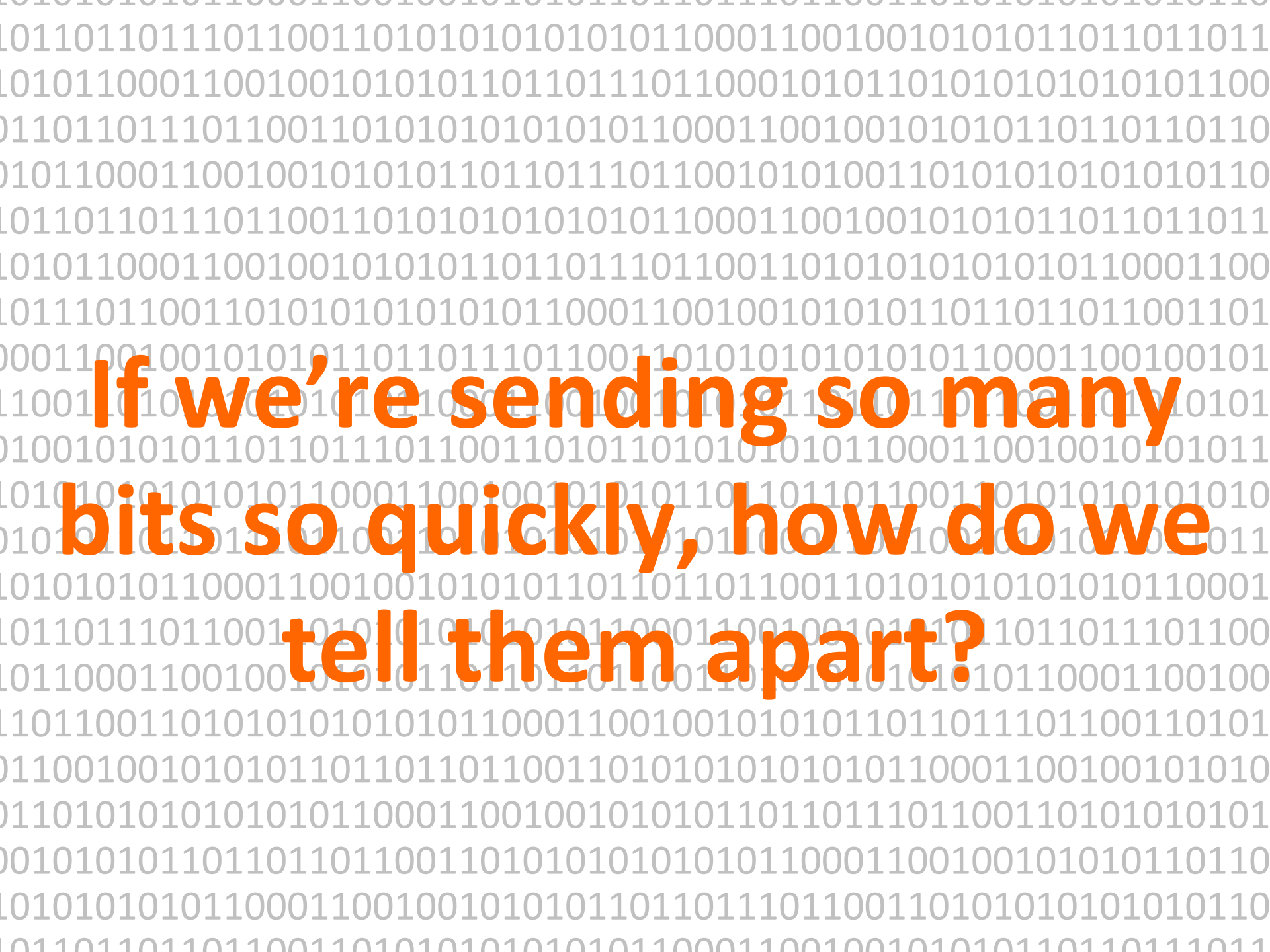


3G and WiFi  
**384 kbps/54 mbps**



r-one robot  
**2 mbps/1.25 kbps**

That's a lot of bits!



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

# That's a lot of bits!

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.



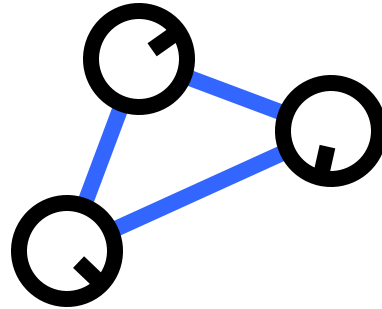
# The nitty-gritty

How do these bits actually get from robot to robot?



# Inter-Robot Communications

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

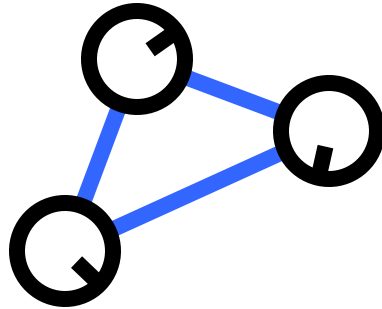


- a. all the time
- b. when they get to their goal positions
- c. only when they need to

## Periodic Communications

Ok, so they need to communicate all the time

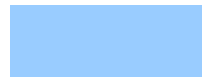
But how frequently?

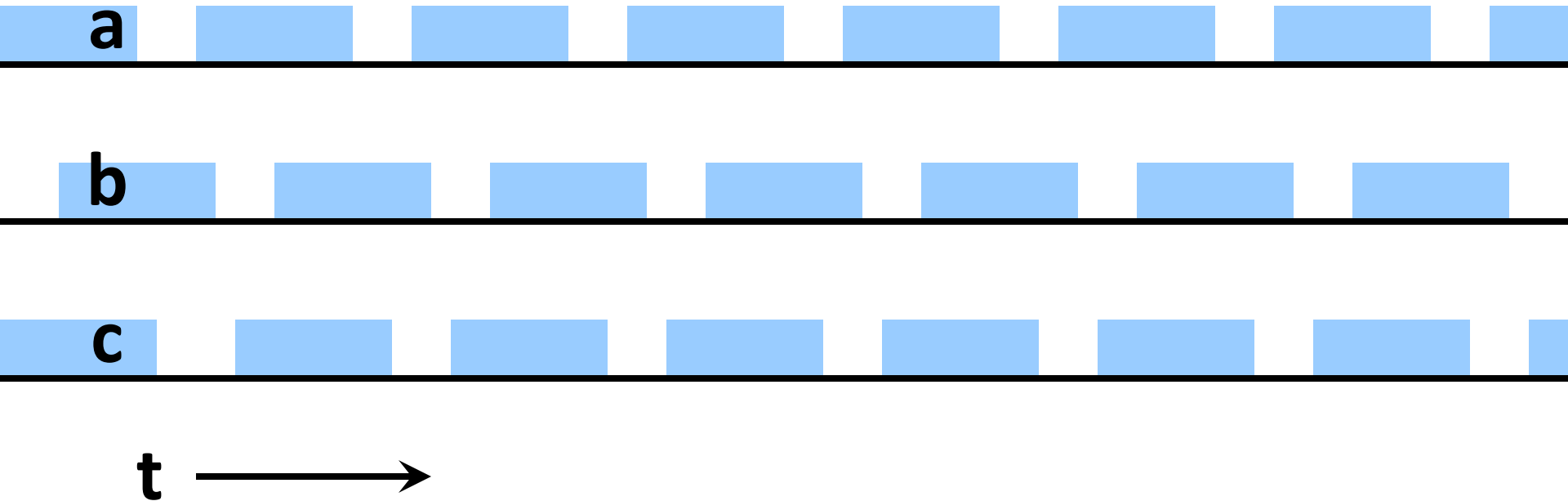
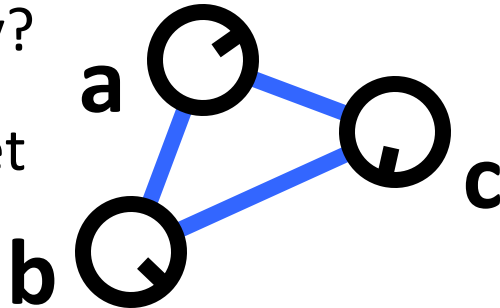


# Periodic Communications

Ok, so they need to communicate all the time

But how frequently?

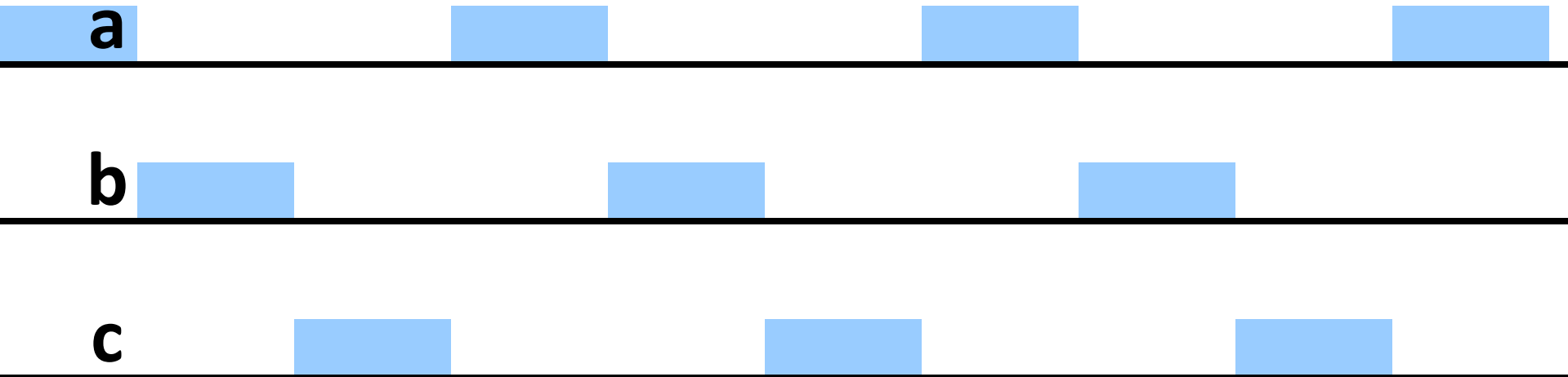
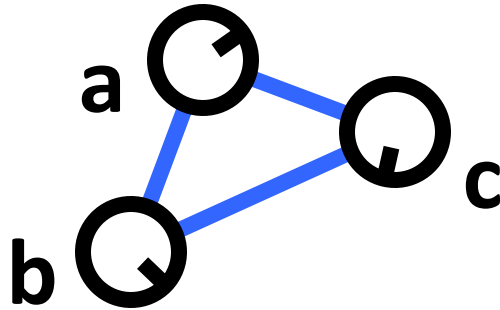
 = one packet



# Periodic Communications

We can avoid collisions with proper spacing

But now what is  
our problem?

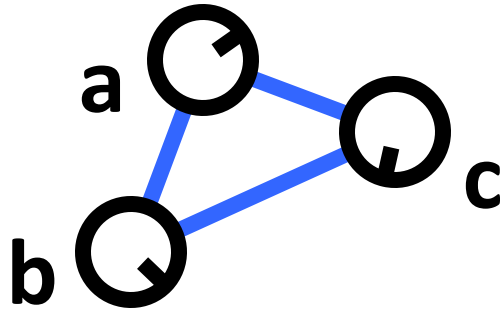


# Periodic Communications

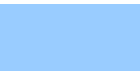
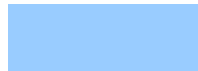
Ok, we'll be more relaxed about the timing.

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

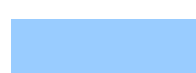
Is this worth the trade-off?



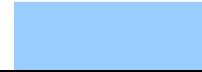
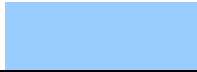
**a**



**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?

# Local Coordinate Systems and a Bearing Motion Controller

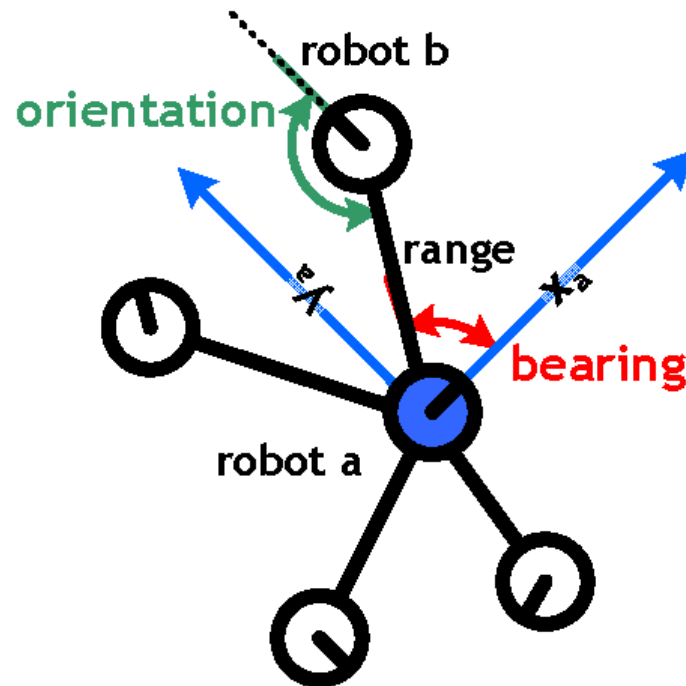
# Local Coordinates

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

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

-or-

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

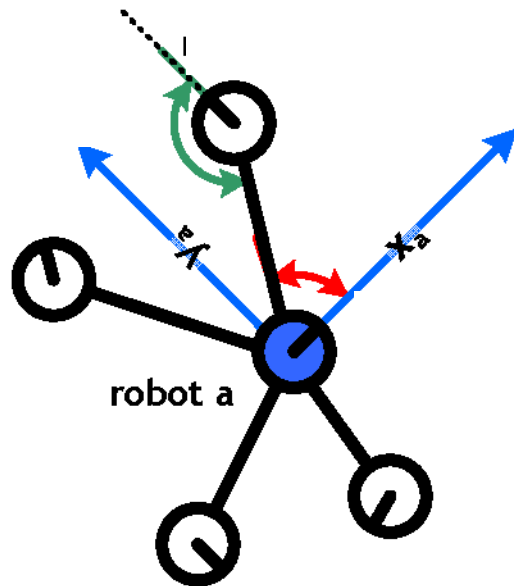


# Motion Control

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

- Translational velocity:  $tv$ , and
- Rotational velocity,  $rv$

How can we compute left and right velocities from  $tv$  and  $rv$ ?



$$v_{\text{left}} = tv - rv$$

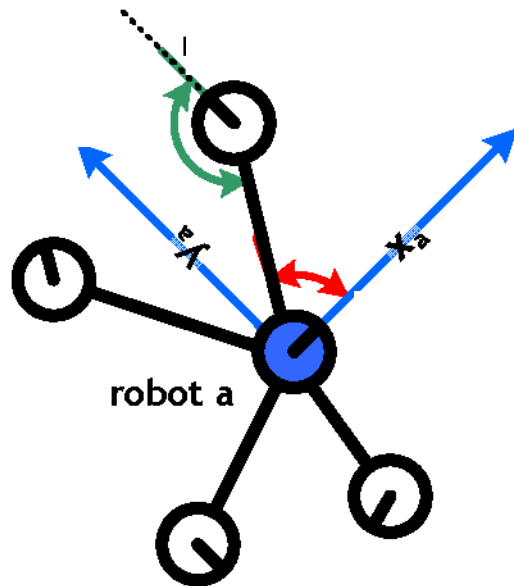
$$v_{\text{right}} = tv + rv$$

# Motion Control

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

- Translational velocity:  $tv$ , and
- Rotational velocity,  $rv$

How can we compute left and right velocities from  $tv$  and  $rv$ ?

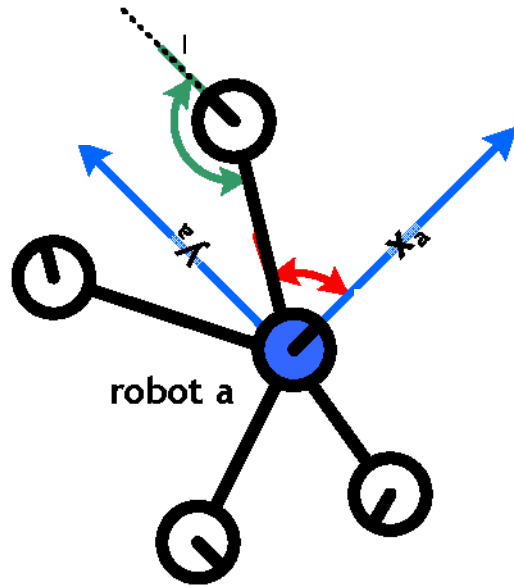


$$v_{\text{left}} = tv - rv * \text{WHEEL\_BASE} / 2$$

$$v_{\text{right}} = tv + rv * \text{WHEEL\_BASE} / 2$$

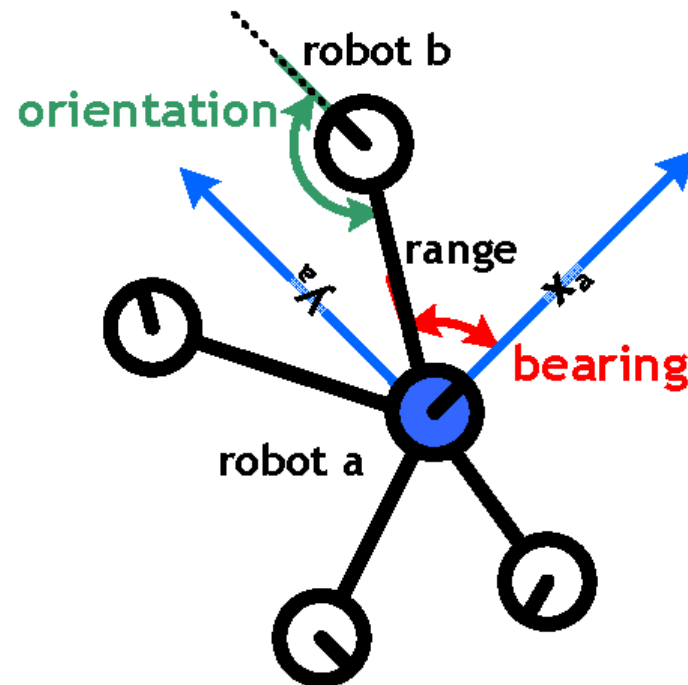
# Motion Control

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



# Local Coordinates: Distance

So what about distance?



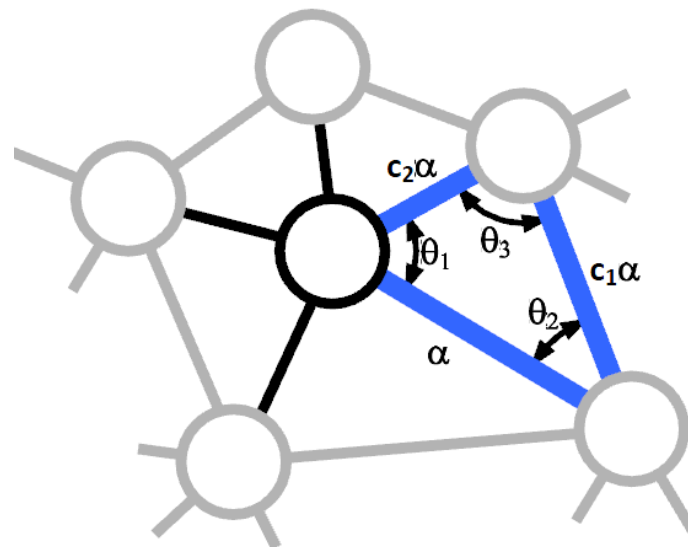
# Scale-Free Coordinates

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,  $\alpha$



a rigid 3-cycle

# Particle Filters

We can produce an estimate of the range over time



# PS07: Follow the Leader



**Follow the leader**

**r-one**

# **Communications**

## **Details**

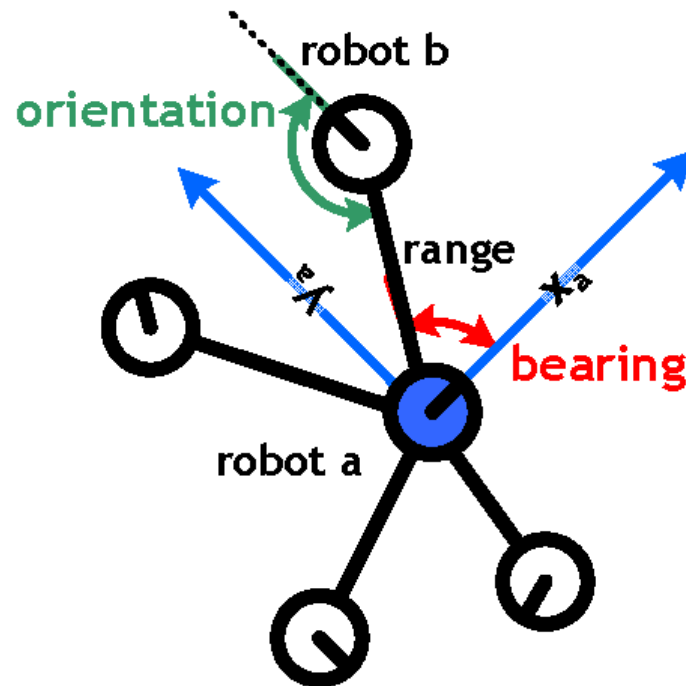
# Local Coordinates

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

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

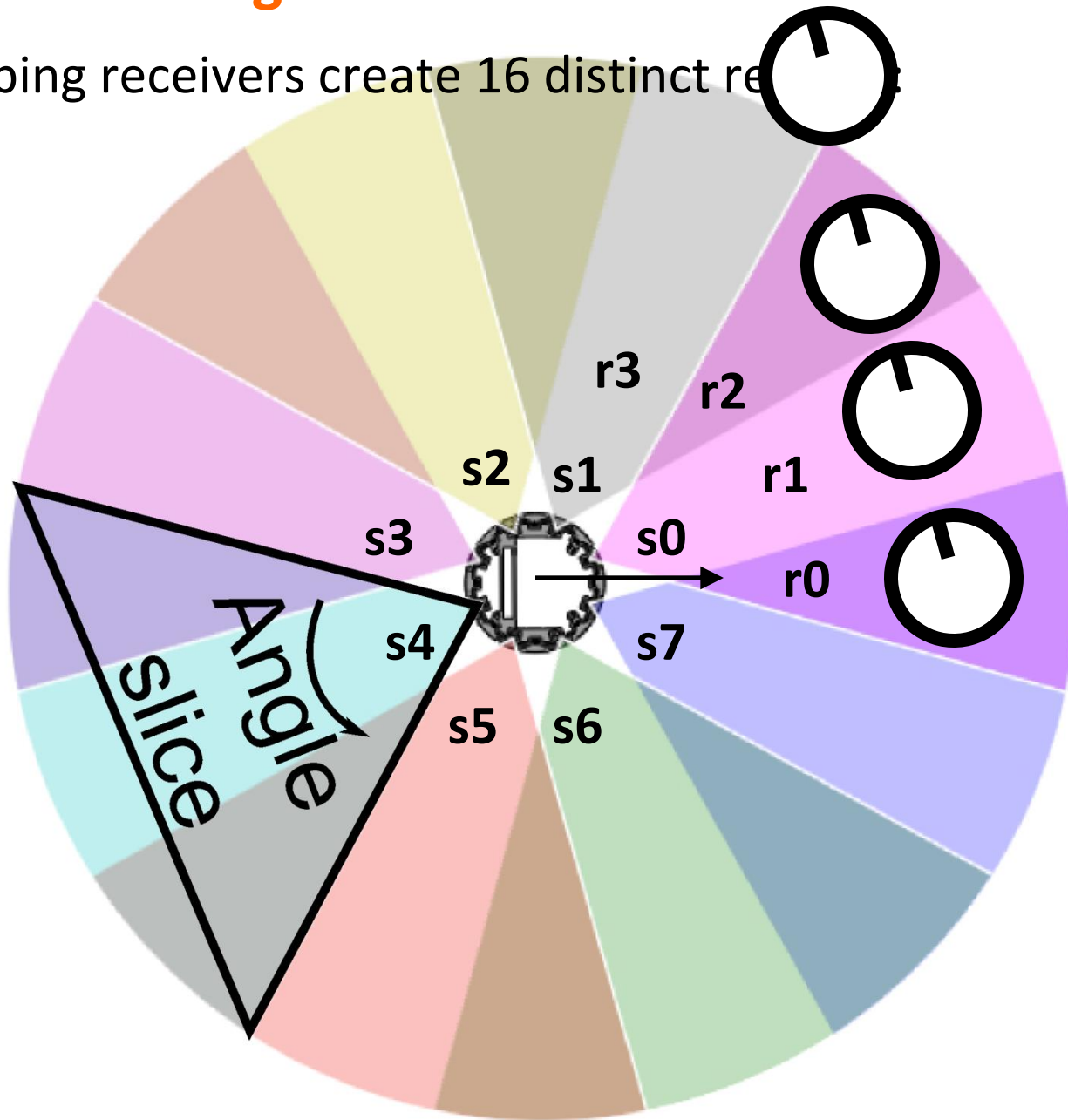
-or-

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



# IR Sectors: Bearing Measurement

8 overlapping receivers create 16 distinct regions:

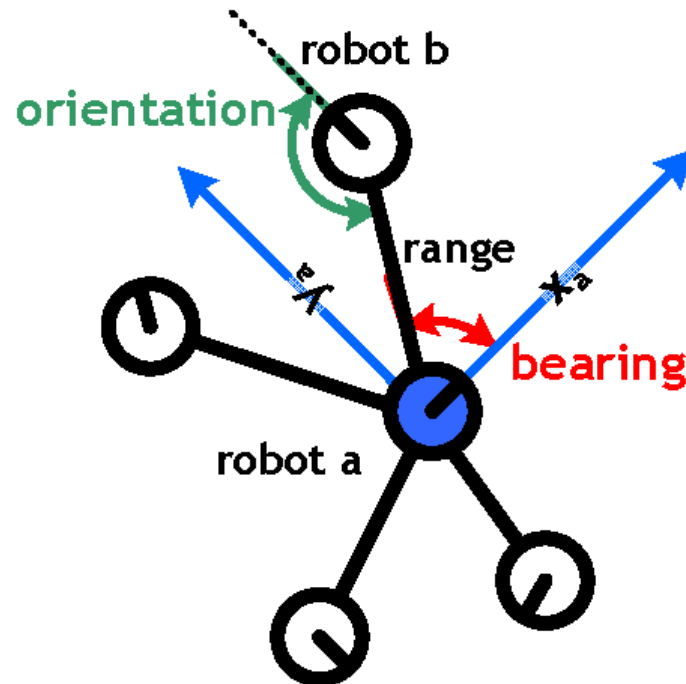


## Local Coordinates: Bearing

We can measure bearing directly

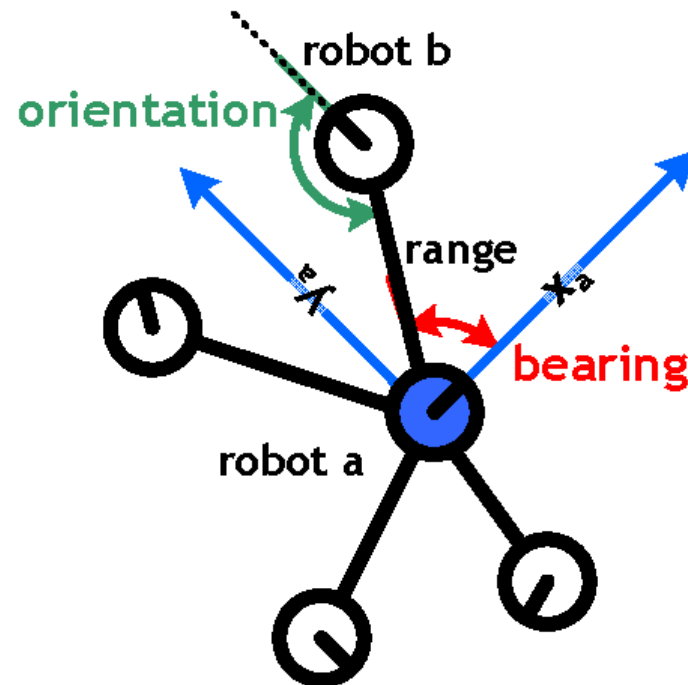
- With an accuracy of around  $\pi/8$

What else can we measure?



## Local Coordinates: Orientation

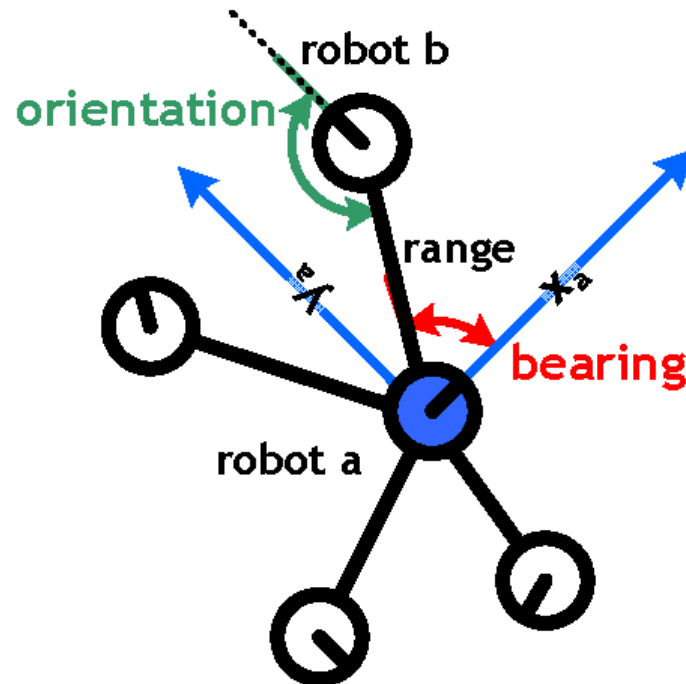
How can we measure the orientation of robot b from robot a's point of view?



## Local Coordinates: Orientation

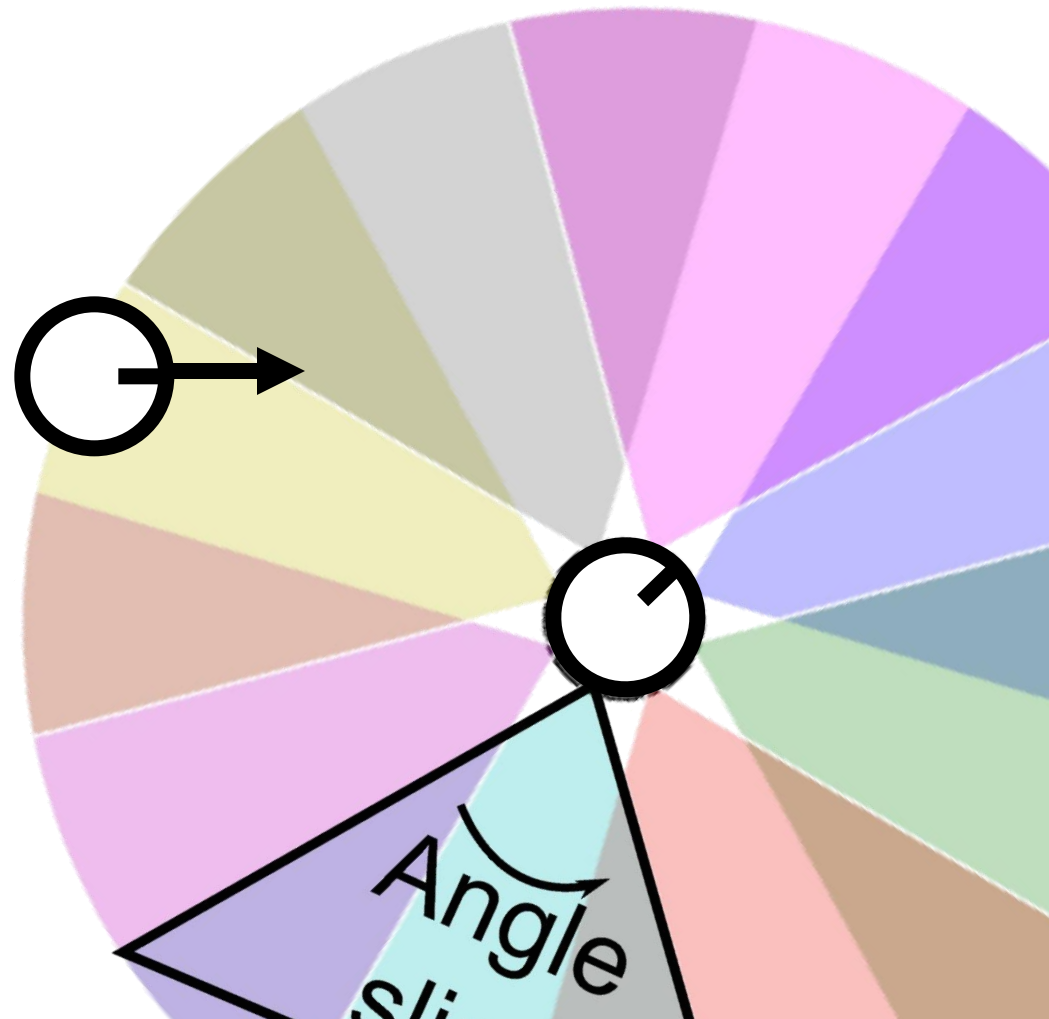
Note that the orientation of robot b from robot a's point of view is the same as the bearing of robot a from robot b's point of view

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



## IR Sectors: Orientation Measurement

8 overlapping *transmitters* also create 16 distinct regions:

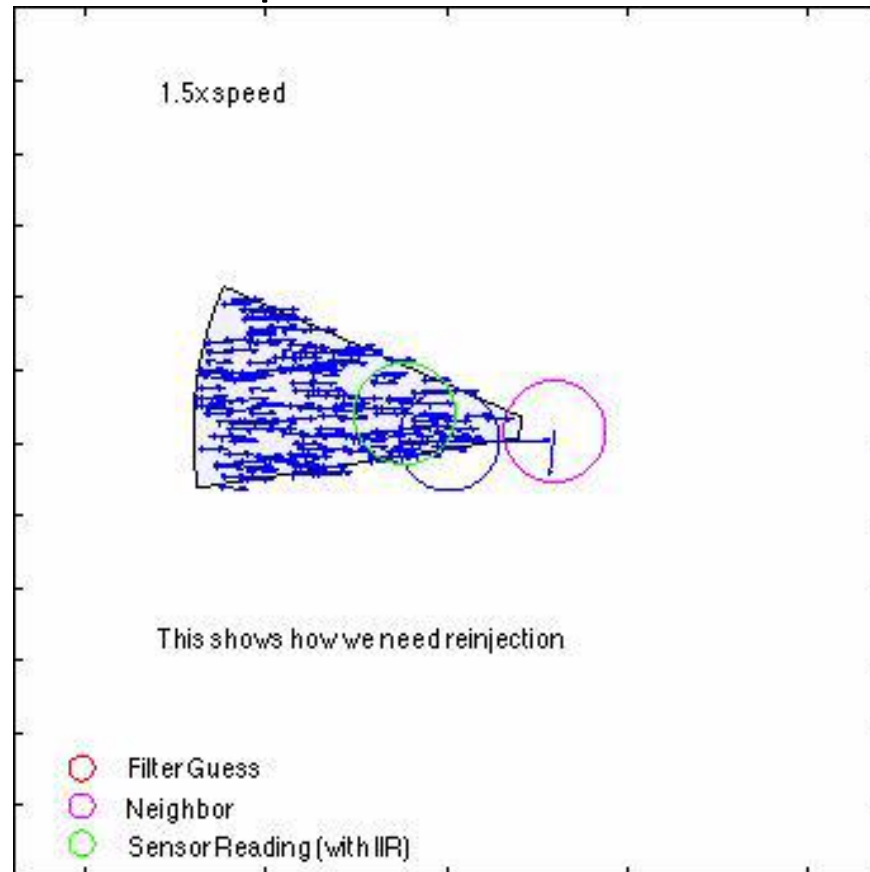


# Local Coordinates: Improving Pose Estimates

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:



**[communications oscilloscope demo]**