ENGR101 T1 2024

AVC Project video processing and movement control

Arthur Roberts

School of Engineering and Computer Science
Victoria University of Wellington

Big picture

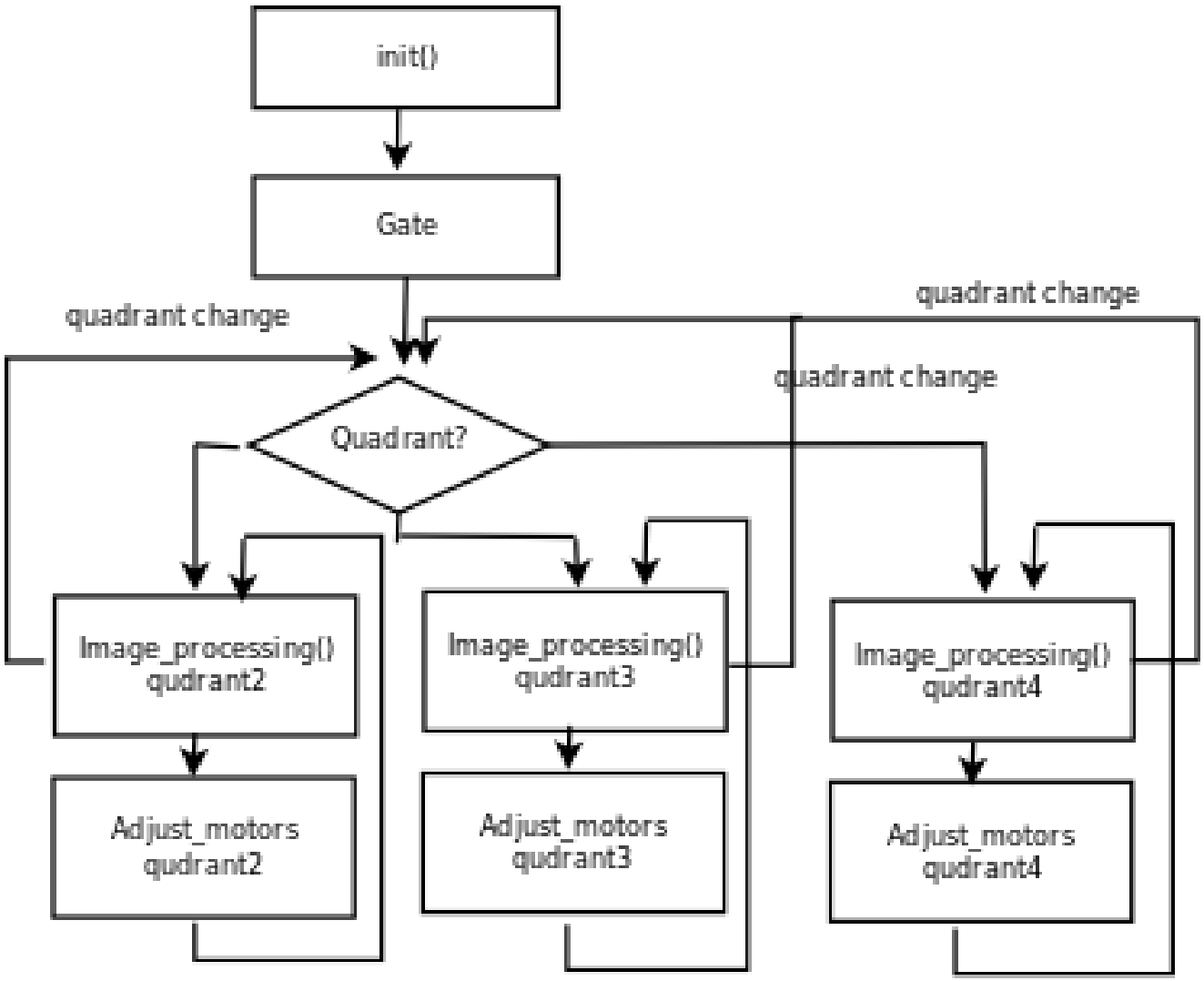

init()
Gate
quadrant change
quadrant change
quadrant change
Quadrant?
Image_processing()
qudrant2
Image_processing()
qudrant3
Image_processing()
qudrant4
Adjust_motors
qudrant2
Adjust_motors
qudrant3
Adjust_motors
qudrant4

Image processing: Quadrant 2 - curved line

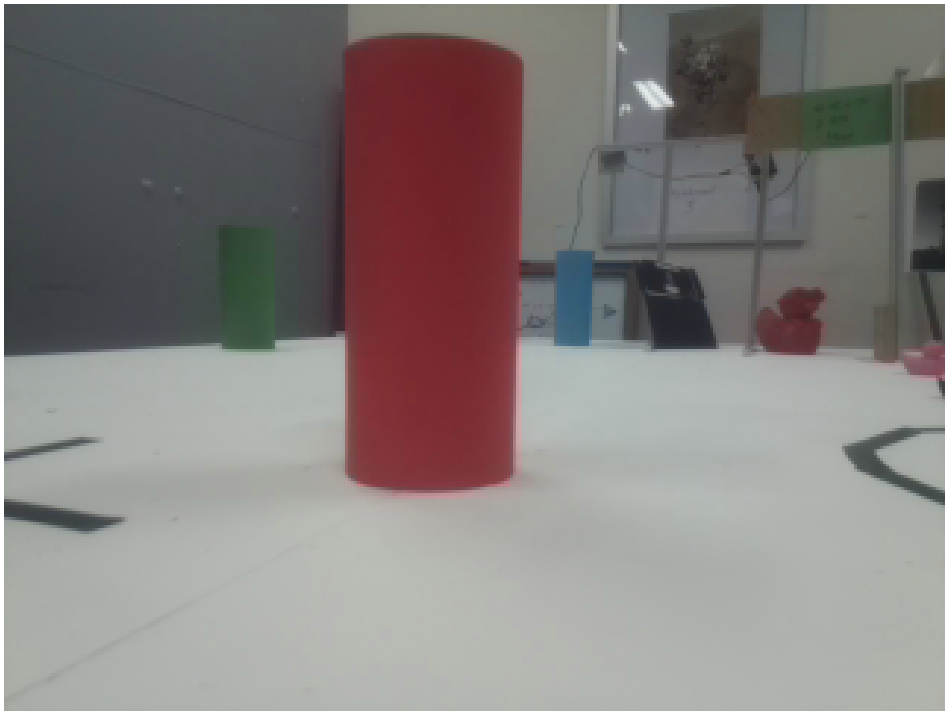
Figure: Robot camera image

Here is the picture from robot camera.
For the robot to follow the line motor speed should be adjusted so that black line is in the middle of the screen

Image processing: Quadrant 3 - intersections

Picture from robot camera. Robot is in Quadrant 3. Select which way to turn.

Image processing: Quadrant 4



Picture from robot camera. Robot is in Quadrant 4. Move towards red cylinder.

Image processing: Quadrant 2 - curved line

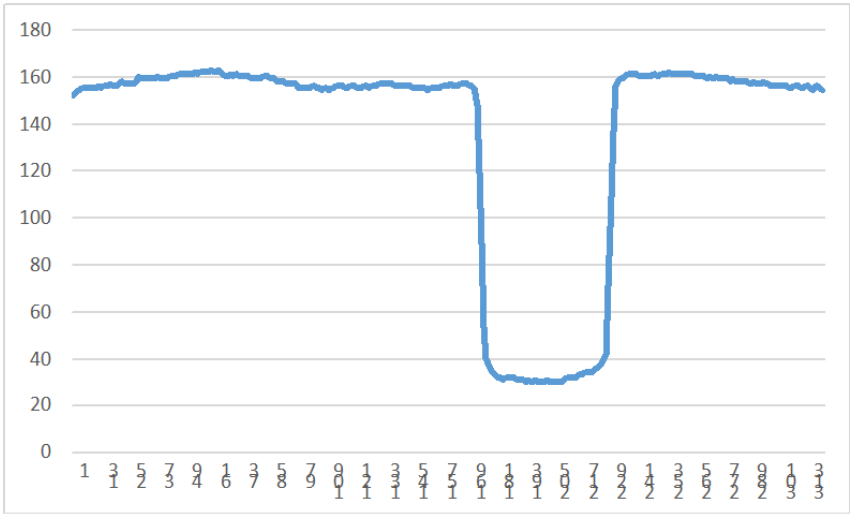
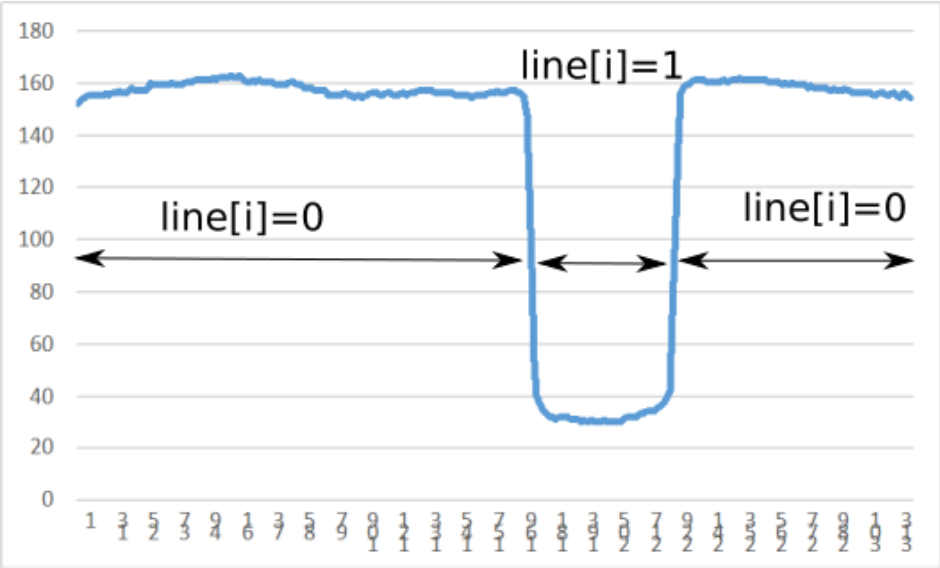


Figure:

Here is the "whiteness" along central horizontal line from previous slide.
Obtained with
get_pixel(row,column,3);
Dip should be in the middle.
Robot should steer in right direction until dip is in the middle.

Keep in mind that depending upon lightning conditions whole picture will drift up (more white) and down (less white). Just like Project2.

Image processing: Quadrant 2



Instead of ..158 169 145 31 32 32...158 169 145

We have ..0 0 0 1 1 1..0 0 0

- It is better to make measurement more clear.
- How to decide which pixel is black and which is white? Some kind of threshold is usually involved: value returned by **get_pixel(row,col,3)** is greater than threshold - pixel is white (wh[]=1). Otherwise, for black pixel - (wh[]=0)
- Value of threshold better be adaptable (changing lightning conditions)

Q2: Array of pixels

We cleared array **line** - it contains only 1 and 0. **Array of black pixels**:

0	0	0	0	0	0	0	0	1	1	0	0	0

1 means that pixel is black - line is there.

And that is *all* we need to generate error.

Error is 0 if line is in the middle of the screen.

Positive if line is to the left of the centre, negative otherwise.

How?

We have indexes for this array. They go

Indexes:

0	1	2	3	4	5	6	7	8	9	10	11	12

One of them is in the middle:

Indexes:

0	1	2	3	4	5	**6**	7	8	9	10	11	12

Lets make new array. Take indexes and subtract middle index from each of them.
In this case middle is number 6.
Indexes:

0	1	2	3	4	5	6	7	8	9	10	11	12

-6

Result is:

Indexes - middle:

-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6

Lets take **array of white pixels** (which contains mostly 0s) and multiply each element of it with **indexes - middle** array:

0	0	0	0	0	0	0	0	1	1	0	0	0
-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
-6*0	-5*0	-4*0	-3*0	-2*0	-1*0	0*0	1*0	2*1	3*1	4*0	5*0	6*0

Result is

0	0	0	0	0	0	0	0	2	3	0	0	0

And finally lets add all elements of last array together:

$$0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 2 + 3 + 0 + 0 + 0 = 5 \qquad (1)$$

What we did is called **scalar product** in calculus. In C++ it is called **inner_product**.

Repeat for other stripe position

Now two pixels before last to the left are white ones:

0	0	0	0	0	0	0	0	0	0	1	1	0
-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
-6*0	-5*0	-4*0	-3*0	-2*0	-1*0	0*0	1*0	2*0	3*0	4*1	5*1	6*0

Adding all elements together:

$$0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 4 + 5 + 0 = 9 \qquad (2)$$

And another...

Now two pixels before last to the left are white ones:

1	1	0	0	0	0	0	0	0	0	0	0	0
-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
-6*1	-5*1	-4*0	-3*0	-2*0	-1*0	0*0	1*0	2*0	3*0	4*0	5*0	6*0

Adding all elements together:

$$-6-5+0+0+0+0+0+0+0+0+0+0+0=-11 \quad (3)$$

Lets have a look what we got here...

0	0	0	0	0	0	0	0	1	1	0	0	0	Result is 5.
0	0	0	0	0	0	0	0	0	0	1	1	0	Result is 9.
1	1	0	0	0	0	0	0	0	0	0	0	0	Result is -11.

Question:

Array of indexes - middle:

-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6

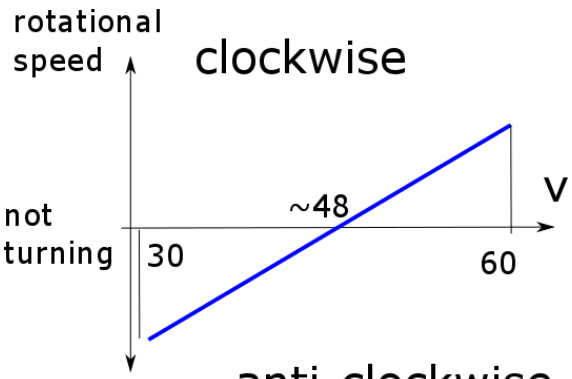
Array of white pixels:

0	0	0	0	0	1	1	0	0	0	0	0	0

What is result (as calculated by algorithm above)?

- That is all about image processing for Quadrant 2.
  We know the error (how far black line is from the centre of the image, in simplest case) and we want to keep this error to the minimum.
- You can easily program algorithm described above.
- Now for **control theory**.
  How to control the motors so that robot movement is fast ans smooth?

Library e101: Motor control




```
set_motor(n,v);
hardware_exchange();
```

- Servo motors (continuous rotation): M1,M4,M5(see board labels)
- Motor number (n) - after M
- **set_motor(n,48);** makes motor stop (or go really slow, adjustable using screw at the back of the motor)
- **set_motor(n,30);** makes motor go maximum speed in one direction
- **set_motor(n,65);** makes motor go maximum speed in opposite direction
- **hardware_exchange();** should be called to update state of the hardware

Left vs Right

One of the motors is left and another is ...

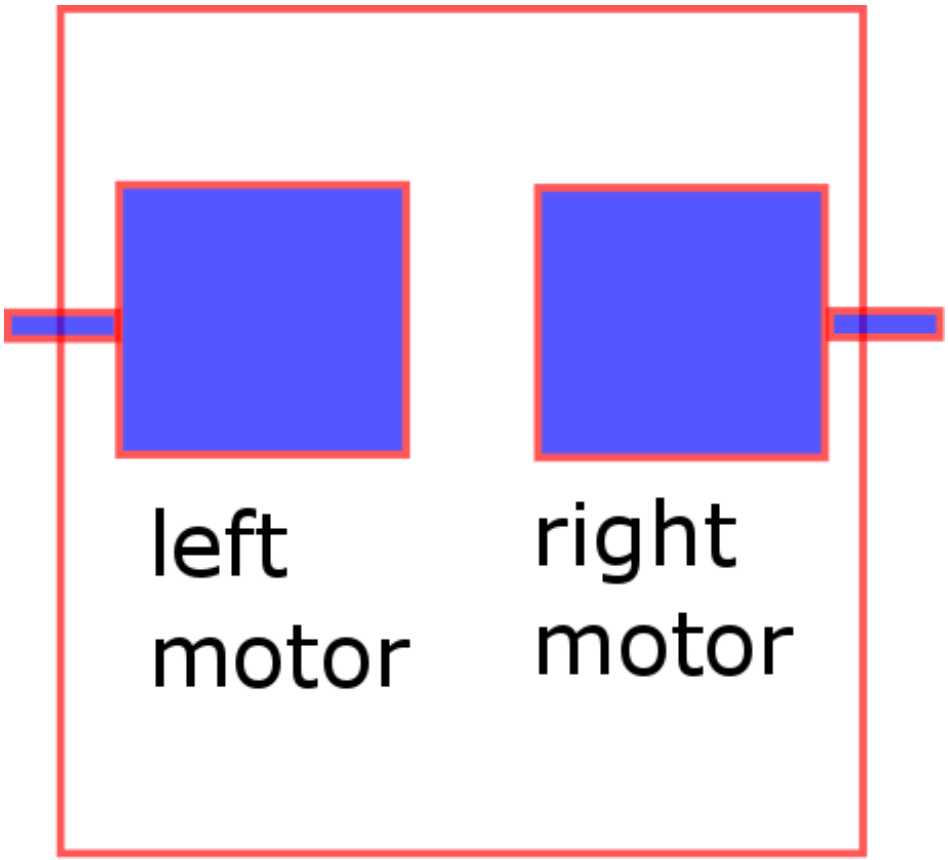


Figure: Left? Right?

- From motor perspective: For robot to move forward motors should be turning in opposite directions
- If passing value **48** stops the motors - to move forward one motor should receive 52 and another 44

Control theory

Control theory describes how to program something to simple to exhibit more complex behaviour...

This requires an **error signal**.

Control theory

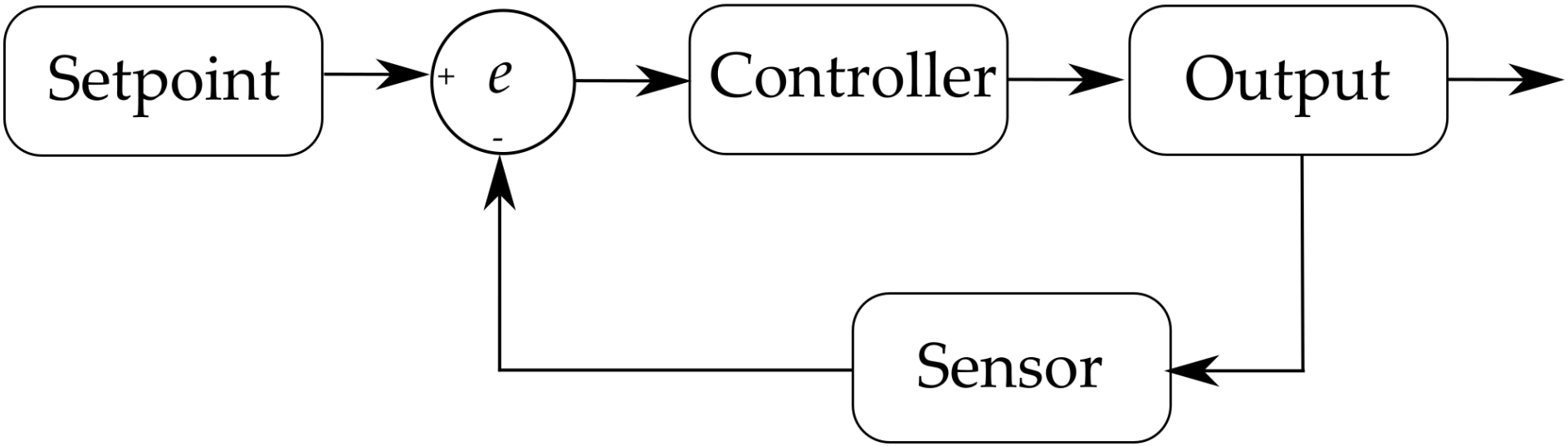
- We want our system in certain state: temperature - 20 degrees, distance to car ahead of us - 5 m, and a lot of other things.
- But it is not.
- Difference is called **error** - and we want to keep it 0
- If error is not 0 - we try to adjust
- Adjustment is constantly repeated

For example, we want to keep landing plane in the middle of runway...
`https://www.youtube.com/watch?v=OoK_dbLDfA0`
Or, less drama, we want black line to be in the middle of the screen.

Diagramatically....

General Control System:

e = calculate error



Example of a **closed loop** system: a system that responds to its current state.

Conditional Solution - simple and WRONG

Easy, is it? What is the problem here?

- if `left half of runway` $->$ full turn right;
- if `right half of runway` $->$ full turn left;
- if `middle` $->$ steady as we go.

Would be fine without any inertia in the system. Light as it is, robot will continue to turn left once it started.
For every complex problem there is an answer that is clear, simple, and wrong.
H. L. Mencken

Read more at: https://www.brainyquote.com

Thermostat

Let's forget about robots and airplanes for a while and consider room heating .

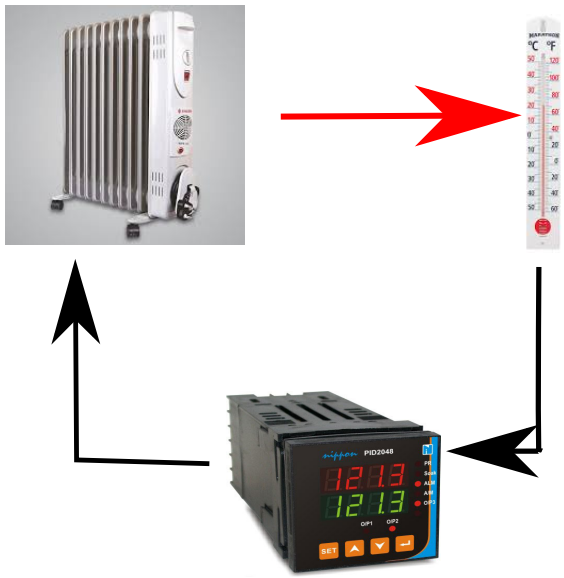
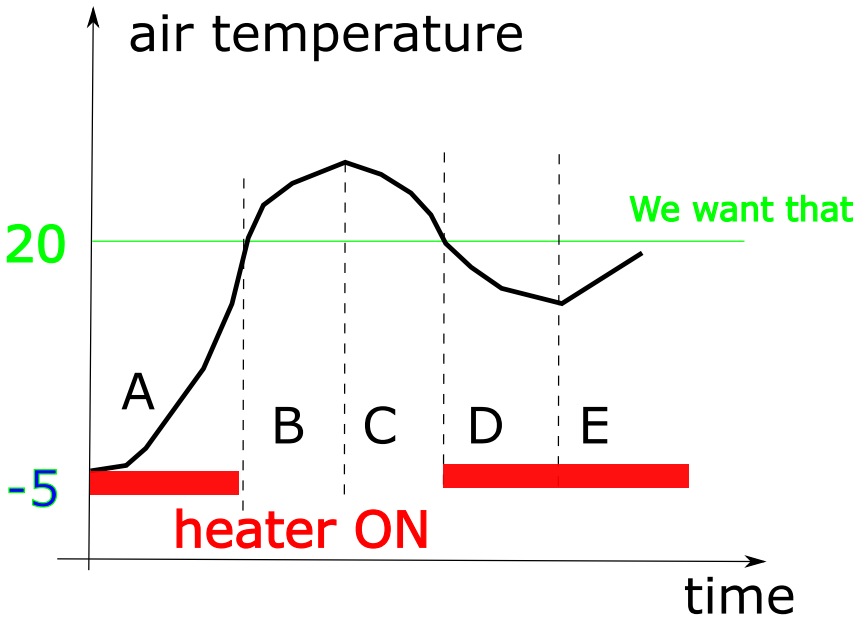


Figure:

- If temperature is lower than comfortable - turn heater ON full blast
- If temperature is higher than comfortable - turn heater OFF

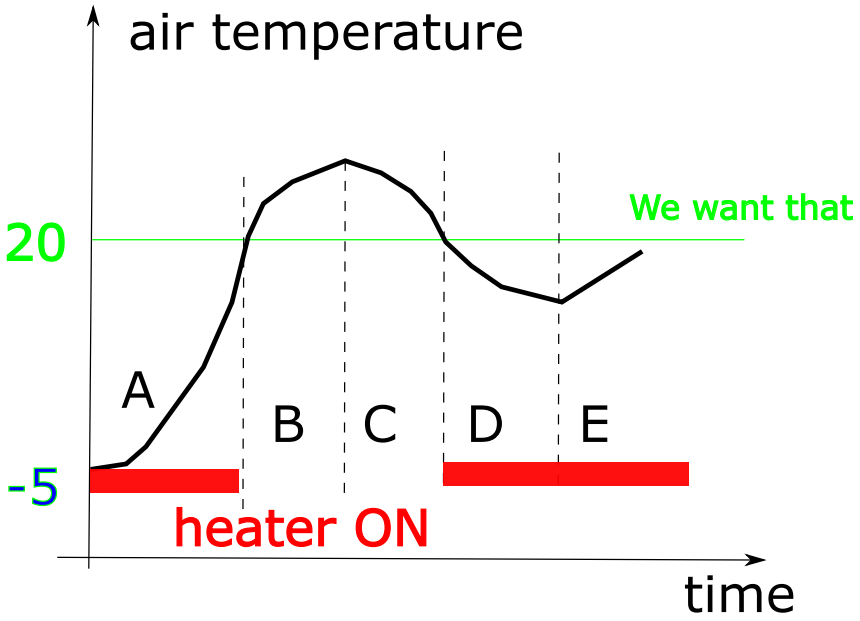
Simple!

What happens?



- A - too cold, turn it ON
- A->B - good now, turn it OFF
- B - heater is OFF but still hot. Temperature in the room keep rising (slower though)
- B->C - temperature of the heater becomes same as air temperature. Air temperature stops increasing.
- C - Heater is OFF, air temperature dropping

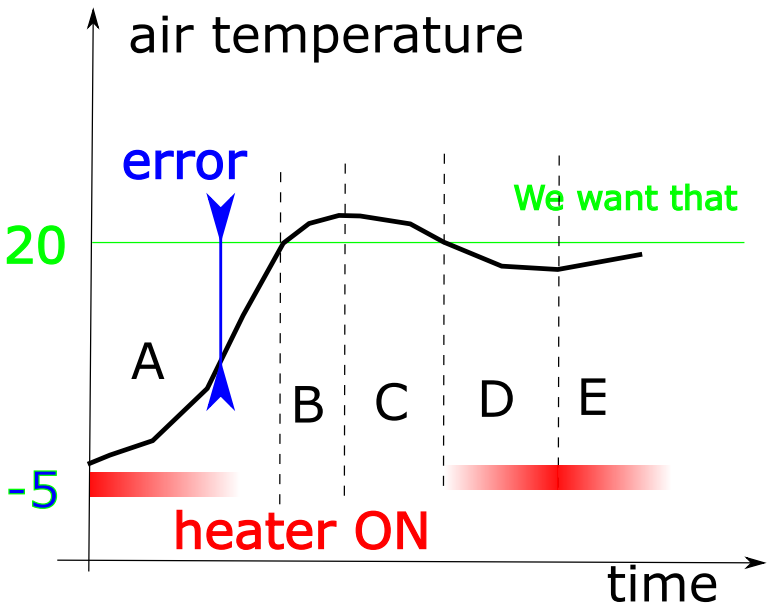
What happens?



- C->D - becoming too cold, turn it ON
- D - temperature of heater rising. But is not high enough to start heating the air yet.

So temperature never actually is what we want it to be but going around and around.
Can we improve on that?

Response is proportional to error



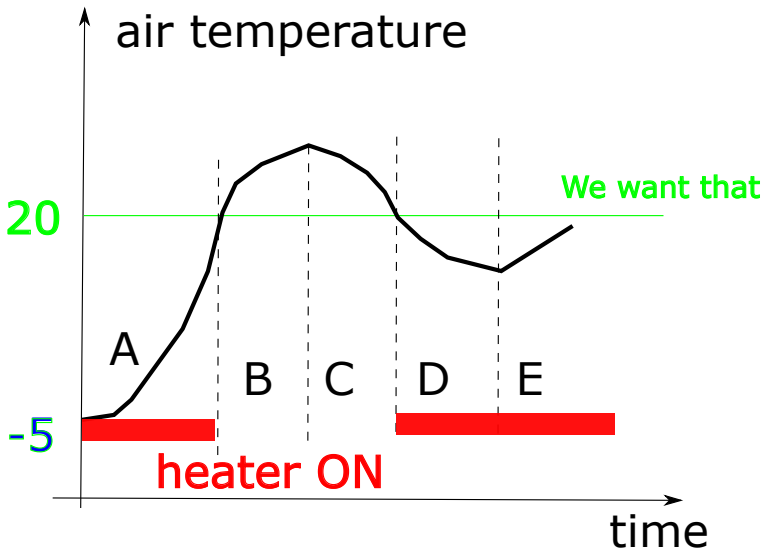
In math form:

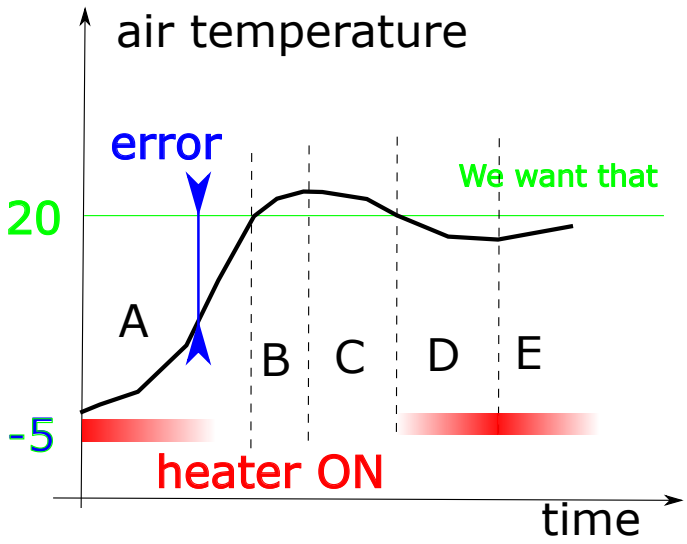
$$heaterSetting = Kp \cdot error \qquad (4)$$

If *error* is big - heat more.
If *error* is 0 - switch OFF.

- Heater have temperature setting
- We measure how far air temperature is off desired value (error) and adjust heater setting accordingly
  - error is big - heater is on HIGH
  - error is small - heater is on LOW
- When time reaches point A->B heater was in LOW setting for a while already, so air temperature in B will not overshoot that much

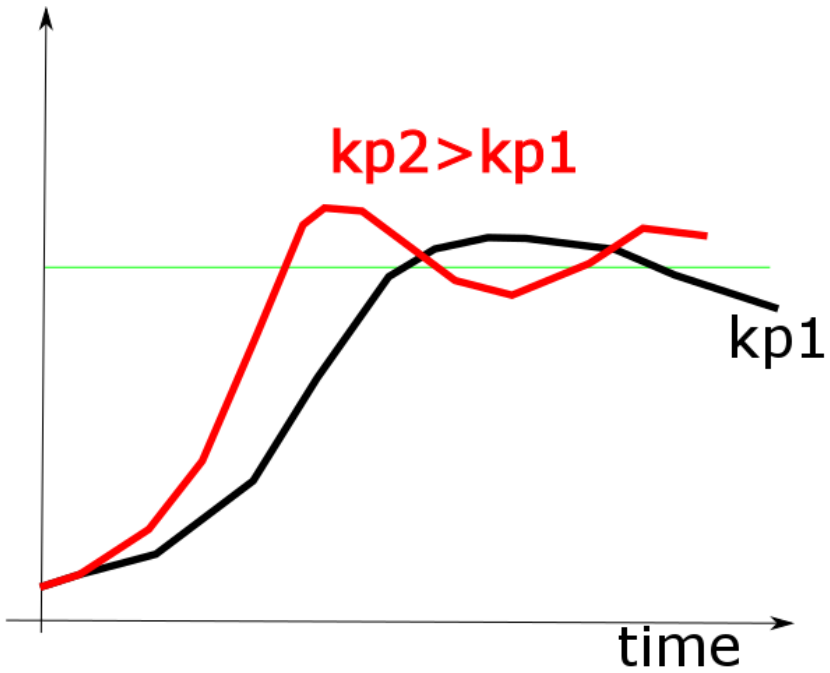
Still it is not quite right...





It takes longer with proportional. It is understandable - controller is not pushing to the limit - its slowing down to avoid the overshooting.
Can we speed things up a bit?

Make it faster...



- It would be good to keep K_p high - faster regulation
- But we risk overshooting

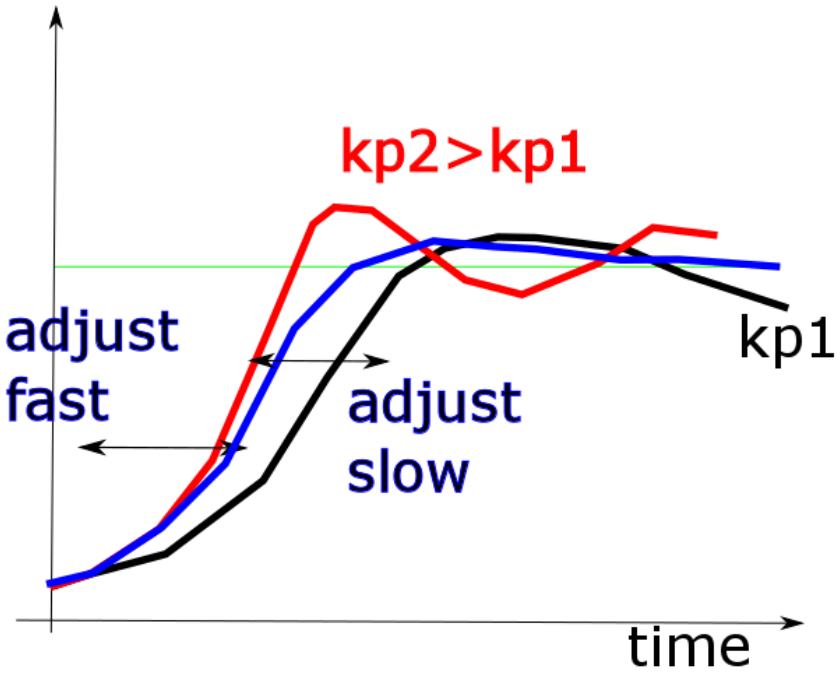
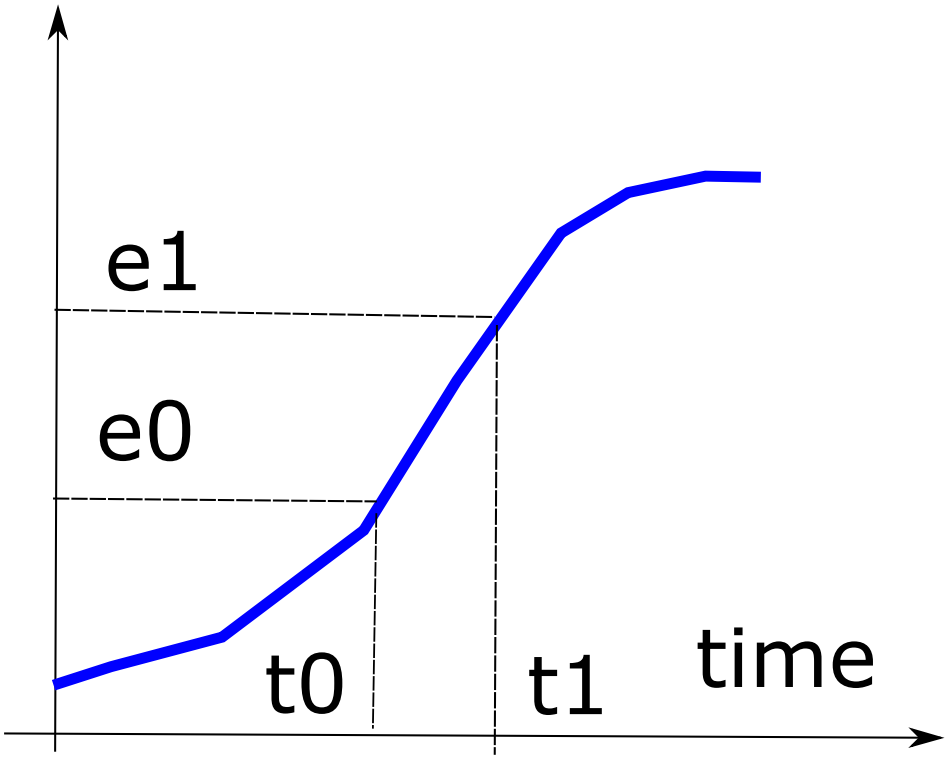


Figure:

- Can we start with big adjustment and then slow down when system approaches desired state?
- How to implement that?
- It is going slow at the start when error is big. Going slow means that rate of change is small.
- It is going faster when error is small and we want to apply the breaks, slow it down.

How detect rate of change?



We measure error at some moment of time.
We measure error again after dt seconds.
Rate of change (derivative) is

$$\frac{de}{dt} = \frac{e1 - e0}{dt} \tag{5}$$

Controller should remember last measurement to calculate rate of change.

We have this equation for the heater:

$$heaterSetting = Kp \cdot error \tag{6}$$

Kp is called proportional coefficient.
In more general terms we can write it as:

$$adjustmnet = Kp \cdot error \tag{7}$$

If we want to slow system down when it approaches 0 error we can add **differential** term:

$$adjustmnet = Kp \cdot error + Kd \cdot \frac{de}{dt} \tag{8}$$

where *Kd* is **derivative** term.

Hm..

All very nice but how it is applicable to the robot?
Robot has to turn to keep black line in the middle of the screen.
If line is not in the middle of the screen - there is an error.

error=-10

turn left
a lot

error=-5

turn left
a bit

error=0

straight

error=5

turn right
a bit

error=10

turn right
a lot

Values of *error* you choose can be different.
How to turn?

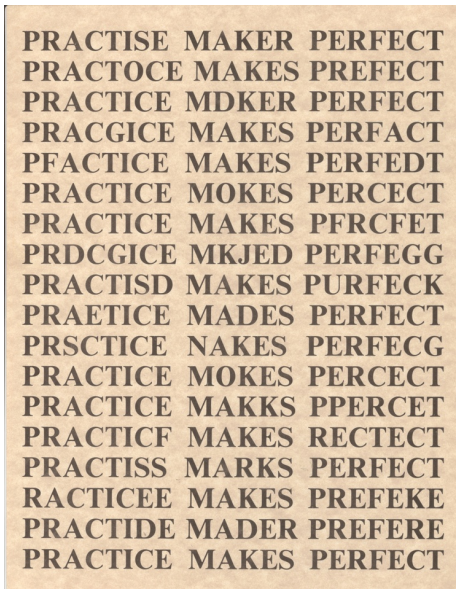
Speed difference proportional to error

- Declare unsigned char v_go_l and v_go_r - speed of left and right motors if robot is to go forward
- Declare unsigned char dv - difference in speed of left and right motors
- Now:
  Speed of right motor: $v_r = v\_go_r + dv$ (set_motor(..,v_l))
  Speed of left motor: $v_l = v\_go_l + dv$(set_motor(..,v_r))
- dv is proportional to *error* (scaling, overflow, type conversions!!!).
  Symbolically, $dv = Kp \cdot error + Kd \cdot \frac{error}{dt}$
- Left, right, forward and backward are confusing things - they change when you connect wires differently.

Demo videos

Bigger Kp results in robot reacting more (turning further) to same error.
Eventually it becomes hysterical.
So answer is - choose right value of *Kp* and it works.

Have to keep trying...



- Set *Kp* and *Kd* both to 0. Robot should go straight(ish - motors are not identical)
- Increase *Kp* until robot follows the line - including curved parts.
- Increase *Kp* until robot starts swinging but still follows the line
- Start changing *Kd* until movement is smooth

Open the gate!

There are 3 functions in E101 library you can use.

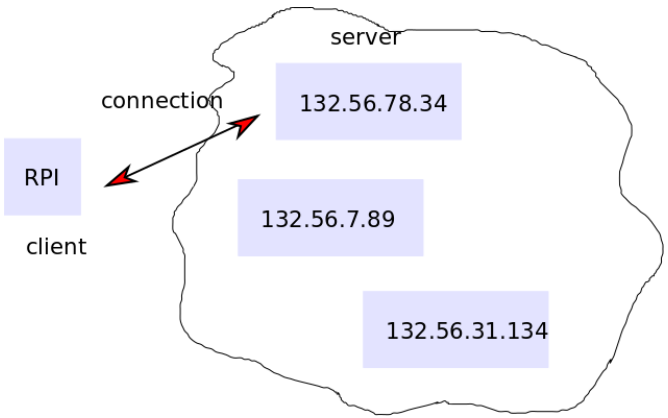
Listing 1: network functions

```
int connect_to_server( char server_addr[15],int port);
int send_to_server(char message[24]);
int receive_from_server(char message[24]);
```

It is about moving around arrays of **char**s and waiting for the server.

Connection

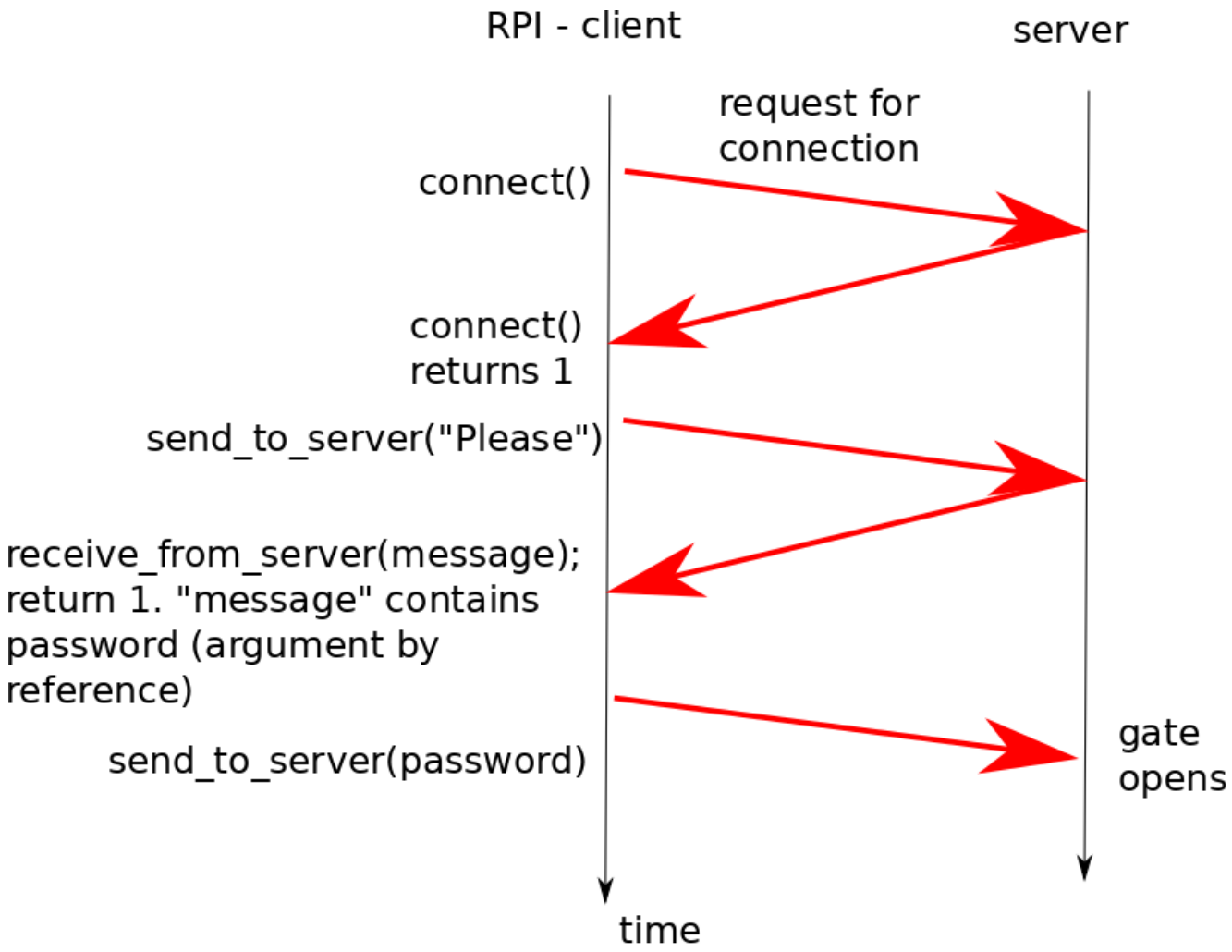
To send information to another computer we have to find it first. We do it by **IP address**.



- int connect_to_server(char server_addr[15],int port);
- We find computer by IP address
- **Port** specifies which program should be used to process the data
- 22(Filezilla) - file transfer protocol, 80 - HTTP (webpage)
- To connect to gate-opening server, use port 1024

Client - server

Two programs exchanging messages.



Questions?