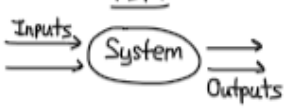
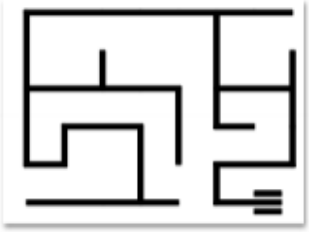
Finite State Machine Example

FSM



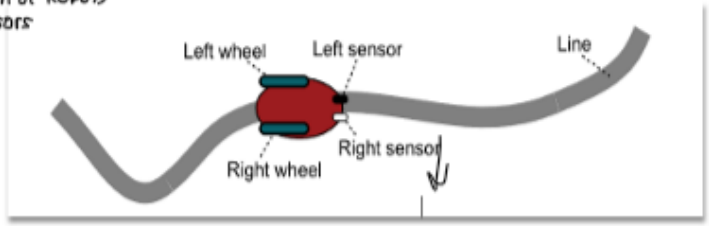
You will learn in this lecture

- Design controller for a line tracking robot
  - Two sensor inputs
  - Two motor outputs



Simple Line Tracker

(Simplified Version of Robot)
– Ours w/ 2 sensors



Example: Turn Right (Gently)

1. Drive the left sensor high
2. Slow down the right wheel w/ the motor by toggling it with a 50% duty cycle

Binary Sensor
On the line → Input = 1
Off the line → Input = 0

Two Sensors	Two Motors (Left Wheel, Right Whe
1,1 on line	1,1 go straight
1,0 off to the right	1,0 turn right
0,1 off to the left	0,1 turn left
0,0 lost – use FSM and previous state to address this issue	

Left, Right Left, Right

Motors
1 → turn motor on
0 → turn motor off

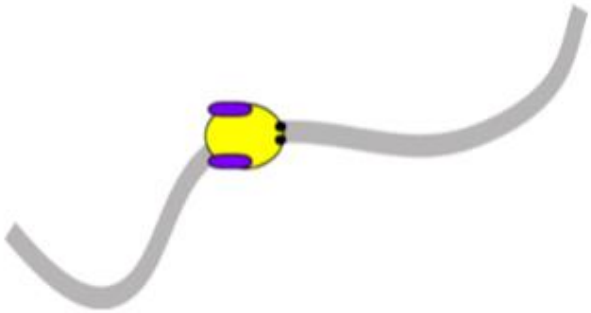
Strategy

To solve FSM,

1. Think of states
   Here, simple
   State
   - Center
   - Left
   - Right

2. What to do about the state I am in?
   - Center → go str

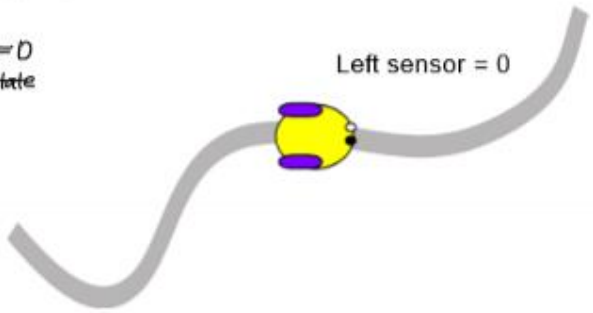
Output
Wait
Input
Next State?



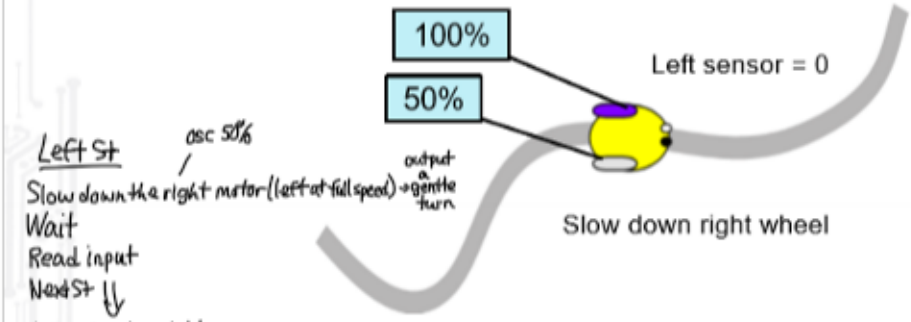
Strategy

Slightly to the left now,
so what happens?

Output w/motors to move str since at center
Wait
Read inputs & see Left Sensor = 0
Next state transition to Left State



Strategy



Left St

asc St6

Slow down the right motor (left at full speed) → output a gentle turn

Wait

Read input

NextSt

keep turning right until back on the line

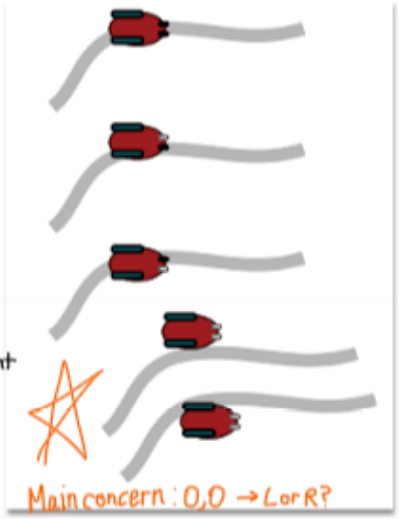
States

Above, we visualized the behavior
So now we should map it out
(state transition graph)

State	Motor
Center	1,1
Left	R L 0,1
Right	R L 1,0

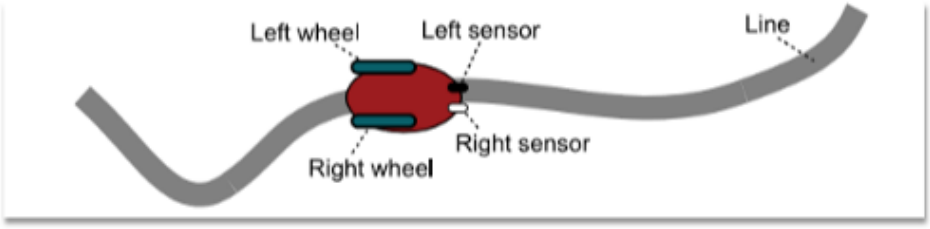
time constant

Motors respond in 100ms,
so run FSM every 50ms



Moore → each state an output

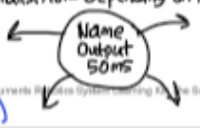
Simple Line Tracker



Two Sensors
1,1 on line
0,1 off to the left
1,0 off to the right
0,0 lost

Left, Right

For us states all need & have:
- Name
- Output
- Delay (50ms)
- state transitions depending on the input



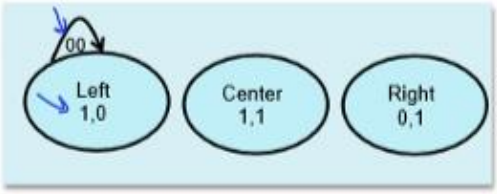
n-bit input → 2^n arrows

Us: 2-bit input → 4 arrows (00, 01, 10, 11)

State Transition Table

Lost → assume you're off to the left

State	Motor	In=0,0	In=0,1	In=1,0	In=1,1
Center	1,1				
Left	1,0	Left			
Right	0,1				

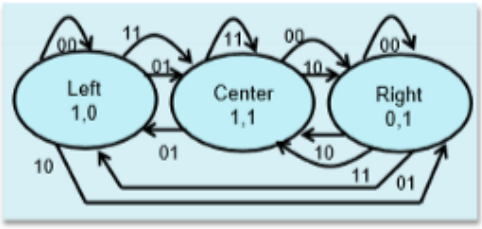



```
State_t fsm[3]={
 {0x03, 1, {  00   01   10   11
 {0x02, 1, { Left,          Center }},
 {0x01, 1, {                Center }},
                            Center }}
};
```

Way off to left, so stop right motor, turn right

State Transition Table – This one combines all of the entries that we have reviewed

State	Motor	In=0,0	In=0,1	In=1,0	In=1,1
Center	1,1	Right	Left	Right	Center
Left	1,0	Left	Center	Right	Center
Right	0,1	Right	Left	Center	Center



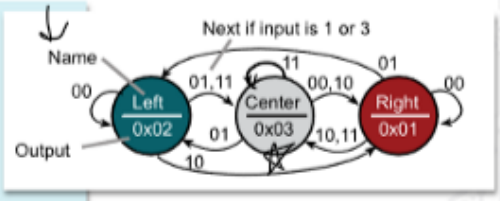

```
State_t fsm[3]={
  {0x03, 1, {Right, Left, Right,   Center }},
  {0x02, 1, {Left, Center, Right,  Center }},
  {0x01, 1, {Right, Left, Center,  Center }}
};
```

Lab: Robot has 8 sensors → more detailed FSM; However, very similar code!

Motors respond in 100ms, so run FSM every ~~100ms~~ 50ms

Robot Implementation

```
struct State {
  uint32_t out;                // 2-bit output
  uint32_t delay;              // time to delay in 1ms
  const struct State *next[4]; // Next if 2-bit input is 0-3
};
typedef const struct State State_t;
#define Center &fsm[0]
#define Left   &fsm[1]
#define Right  &fsm[2]
State_t fsm[3]={
  {0x03, 50, { Right, Left,   Right,  Center }},  // Center
  {0x02, 50, { Left,  Center, Right,  Center }},  // Left
  {0x01, 50, { Right, Left,   Center, Center }}   // Right
};
State_t *Spt;     // pointer to the current state
uint32_t Input;   // 00=off, 01=right, 10=left, 11=on
uint32_t Output;  // 3=straight, 2=turn right, 1=turn left
int main(void){
  Clock_Init48MHz();
  Motor_Stop();  // initialize DC motors
  Spt = Center;
  while(1){
    Output = Spt->out;              // set output from FSM
    Motor_Output(Output);           // do output to two motors
    Clock_Delay1ms(Spt->delay);     // wait
    Input = Reflectance_Center(1000); // read sensors
    Spt = Spt->next[Input];         // next depends on input and state
  }
}
```



12 Motors

13 Timers

9 SysTick

6 GPIO

Summary

- Abstraction
  - Define a problem

    Concepts / principles / processes
  - Separation of policy and mechanisms
    - Interfaces define what it does (policy)
    - Implementations define how it works (mechanisms)
- Finite State Machines
  - Inputs (sensors)
  - Outputs (actuators)
  - Controller
  - State graph
    - States
    - Implementations define how it works (mechanisms)