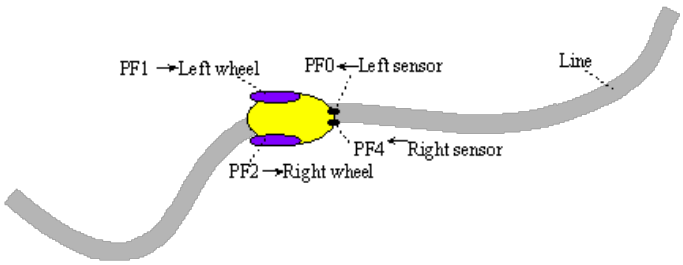
Line tracking robot. The goal is the build a robot that can follow a line. The line is created using reflective tape attached to the floor. The robot has two drive wheels and two line sensors.



LINE TRACKING ROBOT

C10 LineTrackRobot

PROFESSOR JONATHAN VALVANO: Hi.

In this video, let me show you another finite state machine.

We're going to build a line tracking robot.

And this is going to be a Moore finite state machine.

Let's begin with the inputs and outputs.

This line tracking robot has two motors, one on the left and one on the right.

And it has two sensors to tell whether we're on the line.

The goal of this system is to make an autonomous robot which drives along this track.

OK, so I'm going to track this line.

The two motors, if I drive both motors, both the left and the right motor,

they will go straight.

If I drive just the right motor here and not the left, it'll turn left.

And another way around if I drive just the left motor and not the right motor,

it'll turn right.

OK.

So those are my outputs to the two motors.

Let's look at the inputs.

The inputs are infrared sensors.

And these infrared sensors will tell me yes or no

whether or not the two sensors you see positioned here in the front

are on the line or not on the line.

So in this configuration here, if both sensors show one,

that means that both sensors are on the line, which

means the robot is down the middle of the track.

However, if I'm off to the left you see that the left sensor will go dark

and the sensor signal will be 0.

But I'm off just a little bit to the left,

so the right sensor's still on the line.

So the right shows a 1, the left shows a 0,

and that means the robot is off to the left.

If the sensors are reversed, in other words, if the right sensor is dark

and the left sensor sees the line, I'm going to have a 0, 1.

So again, the sensors allow me to see the situation of where the robot is.

I have one more case that I don't know quite how to handle yet

and that is if I'm lost.

And that is both sensors go dark.

I'm either off to the left or the right, I don't know.

And this is where the finite state machine will help us.

If you wanted to build it, here are two motor drivers.

The output of PF2 is going to drive this transistor, driving this right motor.

And the output of PF1 is going to drive this transistor

and drive the left motor.

Again, both motors on go straight.

1, 0 turns left and 0, 1 will turn right.

And here's the sensor interface.

This is a QRB 1134, an infrared sensor with a focal length

of about five millimeters.

This infrared diode emits infrared light.

That light will either reflect or not reflect off the line.

And then that reflected light will either turn on or turn off

this transistor.

This is a thresholding circuit such that this signal here on PF4

will tell me whether or not there's a line.

1 means I see a line.

A logic 0 means I don't see a line.

And I have a second one for the other side.

OK, let's begin with a strategy.

Here I have my robots on the line.

Both sensors show line.

And then the color means here that this dark blue

means these motors are active.

And so both the left and right motors are spinning.

So as it spins forward, we see that the robot goes in a straight line.

But then the robot goes in a straight line, but the line moved.

And now we have something that went wrong.

The left sensor went dark, went 0.

That means I'm off to the left.

So what am I going to do?

My strategy is if I'm off to the left, I want to turn right.

And we saw how to turn right is to slow down the right wheel.

So if I slow down the right wheel and drive the left wheel,

it will turn back towards the line.

So we see that the robot will turn.

The right wheel isn't stopped, it's just running at a slower rate.

And it turns and moves until it's back on the line.

Again, both sensors see the line, both wheels are active,

and so now it's going to move forward again until the line goes away.

The right sensor went dark, left sensor's on.

So what am I going to do?

I'm going to slow down the left wheel, which

will cause it to rotate to turn left.

This line turned a little faster, so it took a couple of times for it

to turn and move until it sees.

So now that once both sensors are activated again,

I'm going to go back to the steady state, which is to drive both wheels so

the robot goes straight.

That's my strategy.

If I'm off to the left, I'm going to turn right and if I'm off to the right,

I'm going to turn right.

And if I'm on the line, I'm going to go straight.

So now what we're going to do is capture that strategy into the state graph.

So we've got to define some states.

So states, if you remember, are what do we know?

What do we believe to be true?

So if I'm on the line, if I'm in the center of the road,

then what I'm going to do is go straight.

And that's what this says.

If I'm in the center, go straight.

If I'm off to the left then I want to turn right.

And if I'm off to the right, I'm going to turn left.

So this is the essence of how I'm going to use

my outputs to build this state graph.

I'm going to use the outputs, which are a

Help

function of the state.

So if I'm off to the left, I'm going to turn right, and if I'm off the right,

I'm going to turn left.

Now, I could've added a lost state.

But we'll see what happens if I get lost when I build the machine.

So now let's build the state transition table.

The state transition table is built in this way.

Down this axis are the state names.

It's a Moore machine.

So every state has a unique output, or every state has an output.

It doesn't have to be unique, but every state has an output.

So that's what I put here.

Now in this state transition table, I need a column for every possible input.

And then what I place in the table is what do I do if I'm in the center state

and if the input is a 1, 1.

The question now becomes what to do next.

And that's what I place in the state.

So let's do a few.

I'm in the center state and I see a 1, 1.

I'm still on the line.

I'm going to remain in the center state.

I've also started to build the graph down here.

And I've also started to build the data structure here just

to show you this one to one mapping.

This entry in the table matches this arrow

in the graph, which matches this entry in the software.

If I'm off to the left, and now I see the line, that means I'm on the line.

So I'll go to the center spot.

If I'm off to the right, and now I see the line, both beautiful lines,

I'm going to go to the center state.

So in this state graph, whenever I see the line

I'm going to see both lines are going to be next into the center state.

That was an easy one.

Let's do a harder one.

What do I do if I'm off to the left?

In other words, the input is at 0, 1.

I'm off to the left.

If I'm in the center state and I just went off to the left,

I'm going to go to the left state.

The other thing I'm going to do, if I'm already in the left state,

I know I'm in the left, and I'm still in the left, what I'm going to do

is actually oscillate here between left and center.

So again, if I'm off to the left I'm going to turn right.

This is where the finite state machine really helps us out.

And that is, what if I go off the track?

Now, what's important to know about a finite state machine

is we know not only what the input is now,

but we know where it was previously.

So what this is telling me is I used to be just a little bit off the left,

and now I'm way off to the left.

You see that?

I used to be just a little bit off, and now I'm way off.

And if so, I'm going to stay in the left state.

So in this mode, I'll stop one motor, drive just the other motor,

and make a hard turn back to the right.

Again, the whole thing is symmetric.

What happened on the left happens on the right.

So this is where I used to be on the line and now I missed the right sensor,

right sensor just went dark, so I'm just a little bit off to the right,

so I'll go to the right state.

And then what will happen is this will oscillate between these two states

while I'm a little bit off to the right, which will again

make a toggling on the left motor and

make a soft left turn back to the line.

So off to the right, I'm going to turn left.

So the red entry in the table matches this arrow,

matches that spot in the software.

This purple entry matches this arrow and that thing.

So again, all three are connected.

Just like going way off to the left, I could go way off to the right.

And you see now the sensor shows 0, 0, but I knew previously

I used to be off to the right.

And so this arrow here will keep me into the right.

So I used to be just a little bit off to the right.

Now I'm way off to right.

And so I'm going to do a complete stop on the left motor,

making a sharp left turn to go back on.

I got a couple of more entries that don't make any sense,

but I've got to put something in there.

This one is weird.

This is telling me I used to be a little bit off to the left

and somehow I jumped to be a little bit off to the right.

That can't happen unless your robot skips.

But I got to put something in there.

Similarly, this one says I used to be off the right

and now I'm off to the left.

I don't know how it happened, but I'm going to put something in here.

This one's a little more plausible.

This one happens if the robot is actually running completely

perpendicular to the track, where it goes from both sensors totally

on to both sensors totally off.

And so in this mode, I'm just going to guess.

I'm just going to pick one and hopefully it was off to the right.

Let's put it all together.

Here's the entire state graph.

Again, one to one mapping between the table, the graph, and the data

structure.

That means no more, no less.

Let's talk about time.

Motors typically run at about a tenth of a second.

So we're going to run our finite state machine

about 10 times faster than that.

So each of my states is going to have a delay of 10 milliseconds.

All right, so now let's do the software.

There's my structure from before.

There's my state and index.

This index is going to be 0 for center, 1 and 2 for the other two states.

I'm going to add some debugging variables, input and output.

Input's obviously going to be the input.

Output's going to be the output.

I'll initialize my GPIO ports, my PLL and SysTick,

and then set the state to assume I'm in the center.

It's a more machine, so what do I do first?

I do the output.

Of course, this is not the output.

This is just reading the output out of the variable

and storing it into this debugging variable.

The actual output occurs when I do the output

to the motor, when I write to port F. And so this variable here

is going to be either a 3, a 2, or a 1 depending upon whether I'm

in the center of the left or right state.

As I mentioned before, each state will wait 10 milliseconds.

So this one is captured out of the structure.

And then I'll wait 10 milliseconds to slow down this driver.

Then I produce my input, which is going to read from the two sensors.

And then it's a Moore machine, so the

next state

is a function of the current state and the input.

So I'm going to go to the next state and then

I'll repeat this over and over again.

Let's show you this in the software.

Here I have the solution to that.

And so let's debug it, I'm running it in the simulator.

So my port F, there's my sensors.

These two bits are my output and I got the Logic Analyzer up,

and so we hit the Go button.

And again, this state here tells me I'm in the center state.

And my two inputs are high, my two outputs are high,

the motor's going straight.

If one of the sensors goes on, we see that when I turn right,

when I'm off to the left, it will turn right by toggling the other motor.

If I get back on the line, I'm going straight again.

If the other sensor's on, I'll toggle the other motor.

So again, I have my inputs and my outputs

and I can see them here in the debugger.

So in summary, what do we have?

This is a Moore finite state machine.

We produce an output, which is a function of the state.

We look at our inputs.

And those inputs tell us what the next state will be.

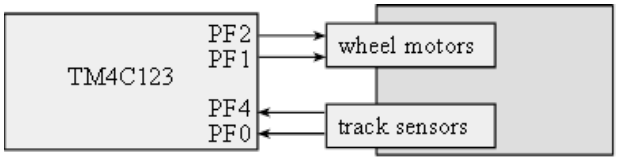
And the idea is to develop a strategy and build

that strategy in to what should the outputs be

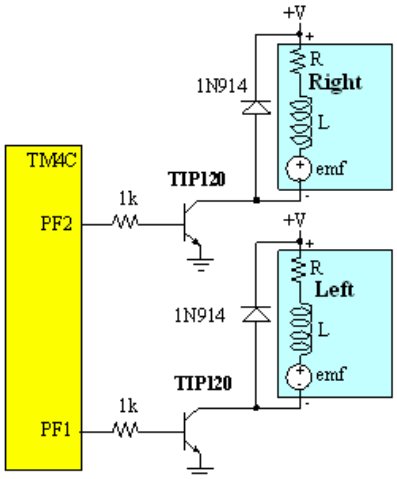
and what should the next states be.

All right, you try.

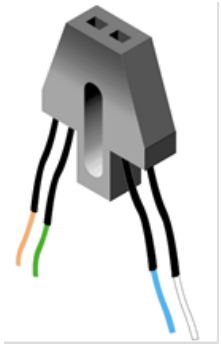
The hardware uses 4 bits of Port F. The initialization will configure Port F bits 2,1 to be outputs and bits 4,0 to be input. These bits correspond to the switch input and LED output of the LaunchPad.

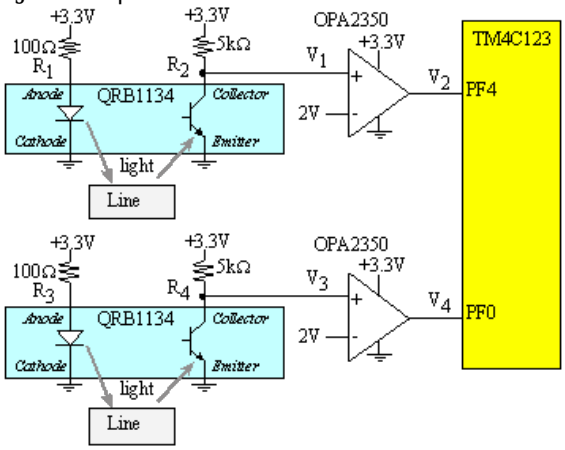


If both motors are on (PF2-1 = 11), the robot goes straight. If just the left motor is on (PF2-1 = 01), the robot will turn right. If just the right motor is on (PF2-1 = 10), the robot will turn left. PF1 drives the left wheel and PF2 drives the right wheel. Each output uses a Darlington transistor to control its motor.

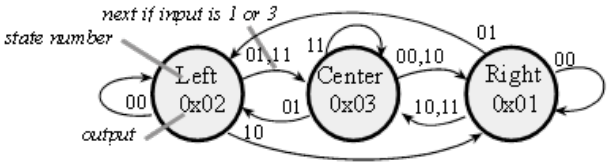


The line sensors can detect whether they see the line. PF4,PF0 equal to 0,0 means we are lost. PF4,PF0 equal to 0,1 means we are off to right. PF4,PF0 equal to 1,0 means we are off to left. PF4,PF0 equal to 1,1 means we are centered on the line. Each input uses an IR reflectance sensor.





The finite state machine implements this line-tracking algorithm. Each state has a 2-bit output value, and four next state pointers. The strategy will be to go straight if we are on the line, turn right if we are off the line to the left, and turn left if we are off the line to the right.



The indexed data structure and main program implement the Moore finite state machine. **Center** is the initial state.

```
void Robot_Init(void);        // Initialize GPIO used for robot
unsigned long Robot_Input(void);    // Input from sensors
void Robot_Output(unsigned long output); // Output to motors
// Linked data structure
struct State {
 unsigned long out;    // 2-bit output
 unsigned long delay;   // time to delay in 10ms
 unsigned long next[4];}; // Next if 2-bit input is 0-3
typedef const struct State StateType;
typedef StateType *StatePtr;
#define Center 0
#define Left 1
#define Right 2
StateType fsm[3]={
 {0x03, 1, { Right, Left,  Right, Center }}, // Center
 {0x02, 1, { Left, Center, Right, Center }}, // Left
 {0x01, 1, { Right, Left,  Center, Center }} // Right
};

unsigned long S;   // index to the current state
unsigned long Input;
```


Help

```
unsigned long Output;
int main(void){
 Robot_Init();
 SysTick_Init(); // Program 10.2
 S = Center;

 while(1){
  Output = fsm[S].out;   // set output from FSM
  Robot_Output(Output);  // do output to two motors
  SysTick_Wait10ms(fsm[S].delay); // wait
  Input = Robot_Input();  // read sensors
  S = fsm[S].next[Input]; // next depends on input and state
 }
}

// ********Robot_Init***************
// Initialize GPIO used for robot
void Robot_Init(void){ volatile unsigned long delay;
 SYSCTL_RCGC2_R |= 0x00000020;  // 1) activate clock for Port F
 delay = SYSCTL_RCGC2_R;     // allow time for clock to start
 GPIO_PORTF_LOCK_R = 0x4C4F434B; // 2) unlock GPIO Port F
 GPIO_PORTF_CR_R = 0x1F;     // allow changes to PF4-0
 GPIO_PORTF_AMSEL_R = 0x00;    // 3) disable analog on PF
 GPIO_PORTF_PCTL_R = 0x00000000; // 4) PCTL GPIO on PF4-0
 GPIO_PORTF_DIR_R = 0x0E;     // 5) PF4,PF0 in, PF3-1 out
 GPIO_PORTF_AFSEL_R = 0x00;    // 6) disable alt funct on PF7-0
 GPIO_PORTF_PUR_R = 0x11;     // enable pull-up on PF0 and PF4
 GPIO_PORTF_DEN_R = 0x1F;     // 7) enable digital I/O on PF4-0
}
// ********Robot_Input***************
// Input from sensors
// 0,0     Lost, off the line
// 0,1     off to right
// 1,0     off to left
// 1,1     on line
#define INPUT (*((volatile unsigned long *)0x40025044))
unsigned long Robot_Input(void){
 // combine PF4 and PF0 into 2 adjacent bits
 return ((INPUT&0x01)+((INPUT>>3)&0x02));
}

// ********Robot_Output***************
// Output to motors
// 1,1 If both motors are on, the robot goes straight.
// 0,1 If just the left motor is on, the robot will turn right.
// 1,0 If just the right motor is on, the robot will turn left.
#define OUTPUT (*((volatile unsigned long *)0x40025018))
```

Help

```
void Robot_Output(unsigned long output){
 OUTPUT = output<<1; // set PF2,PF1
}
```


EdX is a non-profit created by founding partners Harvard and MIT whose mission is to bring the best of higher education to students of all ages anywhere in the world, wherever there is Internet access. EdX's free online MOOCs are interactive and subjects include computer science, public health, and artificial intelligence.