UNIVERSITY OF CALIFORNIA, SANTA CRUZ
BOARD OF STUDIES IN ELECTRICAL AND COMPUTER ENGINEERING

ECE-118(218): INTRODUCTION TO MECHATRONICS

LAB 0 – THE ROACH (FSMS AND HSMS)

PREAMBLE:

Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime[†]. In this class, we are trying to teach you to fish, not giving you fish. That means that there will be quite a bit of flailing around not knowing exactly what you are supposed to do. It also generally means we won't do things for you. This is how you will learn the material, and become better engineers.

OVERVIEW:

In this lab you will program your first robot and solder up a small PCB you will use in later labs. The main purpose of this lab is to get you very familiar with the ES_FRAMEWORK and its debugging tools by coding the behavior of a small two-wheeled robot (the Roach).[‡] You will learn the basics of event checkers, test harnesses, finite state machines, and hierarchical state machines. You will learn (or refresh) basic soldering skills by soldering a small printed circuit board.

COMMENTS:

The labs (like the rest of the class) are time consuming and difficult to some extent. It is very important that you do the work outside of the lab to prepare (readings, prelab work, etc.), or you will spend much more time in the lab. Make sure you start early, and get together with your lab partner so that you can stay ahead of the game. Don't wait until the last minute, and don't attempt the lab cold. Never, ever, attempt the lab without having read through the entire thing front to back (most likely more than once). Most of the information is contained in the supplementary readings; make sure you read those. Getting clarification early on will save you hours of debugging.

The mechatronics class has grown much larger over the past few years. We have been building out the materials as fast as we can (and as budget allows), however we are never going to have enough equipment for everyone. This means you will have to share.

You need to work together to manage the scarce resources that are required for the labs. This means using the sign-up sheets to reserve roaches, being flexible about when you need to solder vs. when you need to program. The lab sections don't necessarily need to be done in order. If you have an Uno32 (either from previous classes or you bought one from Amazon for $30), you can download the firmware and do all of the USE_KEYBOARD_INPUT testing parts on your own. That way, you are ready to load your code onto a roach and test only when you need a physical robot.

The mechatronics labs are open 24 hours/7 days a week. For this lab, all materials are provided in the form of workstations and roach robots for the robot programming part and solder, soldering irons, and mini-beacon solder kits for the soldering part. Get started as soon as possible!

[†] Attributed to Maimonides

[‡] In order to have you concentrate on the software side, we are handing you our own purpose built robot to play with. We know the hardware all works (they have been checked) and we have provided you with a library to access all of the hardware without needing to understand the low-level firmware.

You will spend your first few days working on the prelab. The prelabs are hefty, but they are well worth your time. An extra hour spent on the prelab will save you several hours on the lab itself. You can submit your prelab as early as you like. Be ready to show us your prelab before we check any equipment out to you.

Lab equipment must be reserved by signing your team's name on the wall sheet in BE115. You may not reserve three equipment slots in a row. We should have a roach for each workstation, but in teams of two you will be more than there are workstations, so try to give everyone a chance.

As you progress through the lab, you will need to "check-off" functionality for each part. This means finding a tutor or TA and showing them that your work satisfies the lab requirements. They will electronically check you off. It is STRONGLY recommended that you checkoff each part before moving on to the next part—however, if you cannot find a tutor or TA, there is no penalty for going out of order (just make sure you can "go back" to the working previous section to demonstrate checkoff).[§]

PRELAB:

Choose a partner to do the lab with, and join a group together in the "Lab 0 Group XX" group category on CANVAS. For instructions on how to do this, see the ECE118_LabSubmission document on the website. If you have not chosen a partner by Saturday, we will randomly assign one to you (most likely on Sunday). Note that Piazza is an excellent way to find partners.

Get your mugshot: Visit a TA during lab hours and get your mugshot by Monday at 6pm. This will allow everyone to learn each other's names.

Set up Dropbox: If you don't have one, set up a Dropbox account (https://db.tt/cU0vhjWd). Use selective sync with your dropbox and share a folder for Lab 0 with your labmate. This will allow you to collaborate on your documents and code. It will also make the sync very fast on a new lab workstation.[**]

Take the Class Intake Survey

Each part of the lab has prelab exercises. Complete these by yourself (you are welcome to collaborate with your teammate, but the work should be your own) and submit them using the assignment submission on the CANVAS website. The requirements for the prelab deliverables are detailed in each section, make sure you read the lab carefully and answer them all.

Note that you need to electronically submit your prelab and lab report files in a very specific format. See the ECE118_LabSubmission document on the class website for instructions on how to submit your files and how to verify that you have done so.

[§] This is a place where being organized will help a great deal. Have multiple projects to cover each part you need to checkoff. Keep notes to yourself about what you need to do to set up for checkoff, etc.

[**] We are file-sharing agnostic. You don't need to use DropBox; anything else you can share with your lab partner will be fine. Google drive can work, as can any other file sharing service.

PART 1 – PCB ASSEMBLY AND SOLDERING

OVERVIEW:

This assignment is to assemble and debug one of the supplied minibeacon PCBs. Boards and parts will be distributed in lab after a brief soldering demo. If you are unfamiliar with soldering, make sure you spend some time with one of the students that is good and/or experienced at soldering so they can coach you.

REFERENCE MATERIAL:

- ECE-118 Mini-Beacon documentation
- https://learn.sparkfun.com/tutorials/how-to-solder---through-hole-soldering
  http://www.instructables.com/id/How-to-Solder-Videos%3A-Why-is-soldering-difficult-s/
- Watch the soldering videos in the Tutorials folder on CANVAS (there are six videos here)
- Elecraft solder notes: https://canvas.ucsc.edu/files/5994688/download?download_frd=1

PRELAB:

Watch 3 or 4 videos on soldering technique (yes even if you know how to solder, it's a good refresher). What temperature should you set the iron to? What are the differences in how to handle Tin-Lead Vs Silver solder? How can you tell a hot weld from a cold weld? What does black in the weld mean?

BUMPS AND ROAD HAZARDS:

Soldering irons are very hot and heat up metal and wire quickly. Be careful not to burn yourself; and remember that something very recently soldered is still hot. When soldering diodes, it is extremely important to match the line on the PCB to the line on the diode body. Check 3 or 4 times that you have this right before you solder it down. Silver solder is much harder to work with (but BSOE won't put PbSn solder in the labs).[††] De-soldering is a pain, though it can be done.

OUTLINE:

Start assembly only after becoming comfortable soldering and after examining a completed minibeacon. You will be using this minibeacon in the next lab, so try to solder it well.

Inspect your board carefully. Make sure you note any solder balls, bridged connections, or other things that look wrong. Show your completed board to the TA, Tutor, or Professor for check-off. Power it up and hook it up to an o-scope to verify the frequency (was this what you expected?).

For the lab report, include a picture of the board you soldered, along with any observations you may have had about the experience. You will be doing a great deal of soldering in this class, getting good at it early will help you later.

[††] This is an on-going dispute with EH&S (Environmental Health and Safety) who list lead (Pb) as a neurotoxin (which is most certainly is—if you eat it), and uses the EU's RoHS standards to exclude lead-based solder from the school. That said, the puff of smoke you see when soldering is NOT the lead, but rather the rosin core evaporating. The amount of lead you are exposed to from soldering of any kind is miniscule and presents a vanishingly small hazard. While BSOE cannot furnish the labs with Pb-Sn solder, you can find it very easily on Amazon by searching for "*tin lead solder*." It is cheap, works great, and if you supply your own we won't stop you from using it. You are adults, make your own risk assessments and decide for yourselves.

PART 2 – "HELLO WORLD!" ON A ROACH

OVERVIEW:

This assignment is to get the basic "hello world!" code running on a roach. This will validate the toolchain and ensure that you can get output back from the roach on the serial port.

REFERENCE MATERIAL:

- MPLABX New Project Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language

PRELAB:

None.

BUMPS AND ROAD HAZARDS:

Put roaches up on blocks when on the tables. While the wheels should not spin in this part of the lab, it is essential you prevent them from driving off the lab benches in case you are wrong. Our roaches will NOT survive dropping off the benches onto the floor. They are hand-build, please treat them with respect.

OUTLINE:

Follow the steps in the MPLABXNewProjectInstructions document to get "Hello World!" working on the roach. This will validate that the entire toolchain is working, as is the roach. When you have a problem in the future, this is a good first step to ensure you can program and communicate with your roach.

For the lab report, include a paragraph about getting embedded code up on a new platform.

PART 3 – RUNNING THE ROACH TEST HARNESS

OVERVIEW:

This assignment is to load the roach test harness code onto one of the roaches using the ds30Loader, and verify that the hardware is all working correctly (note that this is an excellent thing to do every time you switch to a new roach).

REFERENCE MATERIAL:

- MPLABX New Project Instructions
- ECE-118 Roach Instructions
- CKO Chapters 1-5.

PRELAB:

None.

BUMPS AND ROAD HAZARDS:

Put roaches up on blocks when on the tables. Make sure your power switch is OFF before you load code. The wheels will turn during this test, ensure you do not drive the roach off the lab bench.

OUTLINE:

Load the test harness .hex file (on the class website) onto your roach using the ds30Loader. Go through the steps in the Test Harness section of the ECE118 Roach Instructions document and ensure that you see the correct behavior. This will validate that the roach hardware is working.

If your roach is not working, bring this to the attention of a TA/Tutor. Put a "squawk tag" on it. **DO NOT** just put it back on the tables.

For the lab report, include a paragraph about testing the roach and what you saw.

PART 4 – ROACH HARDWARE EXPLORATION

OVERVIEW:

This assignment is to write software using the Roach.h/.c module to test and explore the hardware, and get used to reading inputs and commanding outputs on the Roach. Essentially, you are writing your own test harness, or a set of helper functions that allow you to easily interface with the roach.

REFERENCE MATERIAL:

- MPLABX New Project Instructions
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PRELAB:

Create a pseudocode prototype of your test harness. Make sure to describe the tests you will run and what inputs you will use to initiate the tests (bumpers are nice, keyboard inputs also work).

BUMPS AND ROAD HAZARDS:

Put roaches up on blocks when on the tables. Make sure your power switch is OFF before you load code. The wheels **will** turn during this test; ensure you do not drive the roach off the lab bench. Don't simply replicate the test harness we wrote, write your own.

OUTLINE:

Figure out how to test all the different parts of the roach to see for yourself that they work. You should be able to spin each motor forwards and backwards at different speeds, you should be able to read the bumpers (both individually and collectively), read the light sensor, and light the LED bar.

For the lab report, describe your strategy in the design and debugging of this test harness. Describe the difficulties of creating the code. Include any code snippets that are useful and/or demonstrate what you did. Be prepared to show your full code if asked.

Demonstrate your test harness to the tutors/TAs for checkoff. Show them what it does and how it works.

HINT: Develop your code incrementally, get one thing working first, and then verify you have it working, then (and only then) move onto the next function.

PART 5 – EVENT DETECTION

OVERVIEW:

This assignment is to write a set of event detectors which can process bump events and light level detection. Remember that *an event* is a change in something that is detectable (i.e.: INTO_LIGHT). Each event checker is a simple subroutine, written to run very quickly. These very simple event checkers will simply store a sensor reading, and then compare the current reading to a previous reading. If there is a significant difference, the event checker will post an event to the framework. Event checkers of this level of simplicity don't work very well (they tend to "spam," flooding the framework with events).

Event detectors are essential to making embedded projects work well. Mastering them now will help you a great deal in the project.

REFERENCE MATERIAL:

- ECE118 Events and Services Framework
- ECE118 Creating ES_Framework Projects
- ECE118 Debugging ES_Framework Projects
- MPLABX New Project Instructions
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PRELAB:

Create a pseudocode prototype of your event checkers. Make sure to describe the tests you will run to determine that they are working. Include a description of the modifications to ES_Configure.h so that the test harness will run your event checkers.

BUMPS AND ROAD HAZARDS:

Put Roaches up on blocks when on table. They should not be moving, but you don't want to realize that the hard way.

OUTLINE:

You are going to need to build an ES_Framework project and event checkers using the template files in the C:\ECE118 directory. Note that you don't need a full ES_Framework to run your event detectors, and that they should have their own test harness in the event detector code.[‡‡] This is detailed in the documents above. See if you can get the light sensor to spam events. You will be unlikely to get the bump sensors to do so, since they are fairly clean.

For the lab report, describe your strategy in the design and debugging of your event checkers. Describe the difficulties of creating the code. Include any code snippets that are useful and/or demonstrate what you did. Be prepared to show your full code if asked.

Demonstrate your event checkers to the tutors/TAs for checkoff. Show them what it does and how it works.

[‡‡] Use a project-defined pre-processor MACRO to enable the test harness (e.g.: ROACH_EVENT_TEST) with the conditional complication available in C. This means that when you include the .c/.h file in the rest of your code, you won't need to remove the test harness code. But if you want to retest your event checkers, you just load that project and fire it up, and you have your test harnesses. We use this functionality to manage complexity a whole lot.

PART 6 – BETTER EVENT DETECTION

OVERVIEW:

This assignment is to write a better set of event detectors for the bump sensors and the light level detection. In the case of the bump sensors, you are going to implement a simple service to call the event detector periodically (around 200Hz) and debounce the switches. In the case of the light sensor, you are going to implement hysteresis bounds to make sure you don't get spamming of events when the light is very close to the threshold.

You will need to set up a new service in the ES_Framework for the bumper part. This will be your introduction to using simple services, how to post and process events, and using the timers.

REFERENCE MATERIAL:

- ECE118 Events and Services Framework
- ECE118 Creating ES_Framework Projects
- ECE118 Debugging ES_Framework Projects
- MPLABX New Project Instructions
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PRELAB:

Create a pseudocode prototype of your "better" event checkers and the simple service. Make sure to describe the tests you will run to determine that they are working. A description of the modifications to ES_Configure.h so that the test harness will run your event checkers.

BUMPS AND ROAD HAZARDS:

Put Roaches up on blocks when on table. They should not be moving, but you don't want to realize that the hard way.

OUTLINE:

You are going to build an ES_Framework project and event checkers using the template files in the C:\ECE118 directory. Note that you don't need a full ES_Framework to run your light level event detectors, but that you will need a simple service ES_Framework project to see the output of the bump sensors. See the documentation for a tutorial on how to do this. The bump sensor will use the ES_TIMEOUT events to run the service at 200Hz. You will need to initialize the timer and reset it each time.

There are variations on debounce strategies for the switches, any of them can be reasonably implemented. One way to do this is to create an array (or bits in an int) and store past readings. A valid method is to have some number "n" of the switch in the same state to be considered valid. There is always a tradeoff between responsiveness and robustness to bounces (and memory consumed). Arrays, masking, unions, and bitfields are all your friends here. Make sure that your solution is scalable to dealing with all four switches simultaneously (and often many more on your own robot during the project).

For the light sensor, you will be implementing a hysteresis bound for the transitions from light to dark and from dark to light. This makes your event detector robust to small variations in light levels. The "direction" you are going is used to choose the correct threshold. In both your event detectors (service) you are going to need to maintain a history (using static variables) that allows you to notice a detectible change.

For the lab report, describe your strategy in the design and debugging of your event checkers. Describe the difficulties of creating the code. Include snippets of the relevant function (code listing) as part of the lab report. We don't want the entire code base, only the functions you implemented that are relevant.

Demonstrate your event checkers to the tutors/TAs for checkoff. Show them what it does and how it works. Note that you should be demonstrating the light sensor with the test harness, and the bumper using its test harness modifications while running the ES_Framework.

PART 7 – FINITE STATE MACHINE

OVERVIEW:

This assignment is to finally write the finite state machine that implements the behavior of a cockroach on your 'bot. You will do this using all of the code you have already generated, and implementing it using the ES_Framework. The canonical rules of cockroach are: "run from light, hide in darkness, don't get stuck."

Spend time to think through your state machine on this with your partner. A good state machine diagram with states and events well named and labeled will save you hours of lab time.[§§] Test it out with your partner imagining events and inputs and seeing what happens—before you ever code.[***]

REFERENCE MATERIAL:

- HSM Supplementary Lecture Video
- FSM/HSM Toothbrush example
- ECE118 Events and Services Framework
- ECE118 Creating ES_Framework Projects
- ECE118 Debugging ES_Framework Projects
- MPLABX New Project Instructions
- ECE118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PRELAB:

Create a good drawing of your FSM with all states and transitions labeled. Neatly done on paper is very good, there are also various programs that can be used to draw this (though really do start with pen and paper). GVedit and Umlet on the lab computers, Draw.io online has also been used in the past.

Create a list of the helper functions you think you will need, with a brief explanation of what they can do (take a look at Roach.h for an example of how we do it).

BUMPS AND ROAD HAZARDS:

Put Roaches up on blocks when on table. Do **NOT** ever run your roach live on the lab benches. If they are moving, they go on the floor. We don't trust you to catch them when they roll off the benches.

OUTLINE:

[§§] Not to belabor the point, but think a **3-5x** improvement in your lab efficiency.

[***] This is "walking through your code," and is incredibly useful to finding the corner cases and missing state/event pairs that will make debugging difficult later. As you will discover later, the longer a bug exists in the code, the more difficult it is to remove it. Here you have the opportunity to remove the bug before you have even written a single line of code!

You are going to set up a new ES_Framework project and use the FSM templates to get a simple state machine to work. You have already familiarized yourself with the roach hardware (in Part 4), and written your event detectors (in Parts 5/6), so all that is left is to code up the behavior.

You will link to the code you wrote in these separate projects, not copy them into your new project.[†††] This is going to be a good practice going forward.

Using your very well drawn state machine diagram, you are going to implement the roach behavior. Use the Keyboard Input and the TattleTale to initially test your state machine, then turn off the Keyboard Inputs and stimulate the roach. The challenge is to understand what state/event pairs are coming, and where they are going. The TattleTale will help with this a great deal.

Develop incrementally as it is much faster in both the medium and long term. It will be very useful to create "helper" functions such as goForward(int x) and turnHalfSpeed(int x) so your state machine is easy to read and easy to code. Give some thought to what helper functions you will need.

Make your roach hide in darkness, and run from light. Have it be responsive to bumps when running from the light (but not when in darkness).

For the lab report, include your updated FSM diagram, describe your strategy in the design and debugging of your FSM. Describe the difficulties of creating the code. Include the code listing of relevant parts of your code in the lab report. Include a link to video of your working robot.

Demonstrate your FSM to the tutors/TAs for checkoff. Show them what it does and how it works. Make sure your roach can meet the specifications and react robustly to changes in the environment.

PART 8 – HIERARCHICAL STATE MACHINE

OVERVIEW:

Now that you have the basic functions down pat, see if you can make it more interesting to watch by endowing the machine with a more complex behavior. The basic rules of cockroach still apply: run from light, hide in darkness, and don't get stuck.

You are going to use a Hierarchical State Machine (HSM) to implement this complicated behavior. Simply put, an HSM is just like a flat state machine, but states can have their own state machine within them. The way this works is that an event is passed down to the lowest level that it can go, and if the event is handled at the lower level, the sub-state machine consumes the event and returns ES_NO_EVENT to the super-state machine. If the event is unhandled at the lower state machine then it is passed back up and handled by the super-state. This allows for common reactions to an event to all be handled in one place only, but more specific reactions to be implemented as well (for CS geeks among you, this is reactive behavior equivalent to polymorphism).

The new rules of cockroach are:

- While hiding in dark, if you get bumped, move away from the bump and keep moving for at least ½ second after you've released the bumper.
- While running in light, periodically (every 5-10 seconds), dance a jig or implement an active search for darkness. Be creative here.

[†††] Again, this is why we conditionally compile the test harnesses with a project-scope MACRO.

- Make sure your "don't get stuck" is robust to various scenarios, so that the roach really does not get stuck.

Again, spend time to think through your state machines on this with your partner. In this case there will be a top level state machine (relatively simple) and each of those states will have a sub-state machine. A good state machine diagram with states and events well named and labeled with save you hours of lab time. Test it out with each other by imagining events and inputs and seeing what happens—before you ever code. Doing this will catch errors early and save you lots of time in lab.

REFERENCE MATERIAL:

- HSM Supplementary Lecture Video
- FSM/HSM Toothbrush example
- ECE118 Events and Services Framework
- ECE118 Creating ES_Framework Projects
- ECE118 Debugging ES_Framework Projects
- MPLABX New Project Instructions
- ECE 118 Roach Instructions
- Essential C – if you are unfamiliar or rusty with the C programming language
- CKO Chapters 1-5.

PRELAB:

Create a good drawing of your HSM with all states and transitions labeled. Include drawings of all sub-state machines and good notation as to where they are called from.

BUMPS AND ROAD HAZARDS:

Put Roaches up on blocks when on table. Do **NOT** ever run your roach live on the lab benches. If they are moving, they go on the floor.

OUTLINE:

You are going to set up a new ES_Framework project and use the HSM templates to get a simple hierarchical state machine to work.

Your roach should be reactive to bumps in both light and dark. It should also implement some strategy to actively search for darkness or do some dance periodically. HSMs are a very good way to manage the complexity of a difficult reactive task, and learning to do them well here will serve you well later in the project (typical 118 projects wind up with HSMs 3-4 levels deep).

For the lab report, include your updated HSM diagrams, describe your strategy in the design and debugging of your HSM. Describe the difficulties of creating the code. Include the code listing of relevant parts of the code in the lab report. Include a link to video of your working robot.

Demonstrate your HSM to the tutors/TAs for checkoff. Show them what it does and how it works. Make sure your roach can meet the new specifications.

CHECKOFF AND TIME TRACKING

Student Name:_____ CruzID_____@ucsc.edu

Time Spent out of Lab	Time Spent in Lab	Lab Part - Description
		Part 1 – PCB Assembly and Soldering
		Part 2 – "Hello World!" on a Roach
		Part 3 – Running the Roach Test Harness
		Part 4 – Roach Hardware Exploration
		Part 5 – Event Detection
		Part 6 – Better Event Detection
		Part 7 – Finite State Machine (FSM)
		Part 8 – Hierarchical State Machine (HSM)

Checkoff: TA/Tutor Initials	Lab Part - Description
	PreLab – Preparation for the Roach Lab
	Part 1 – PCB Assembly and Soldering
	Part 4 – Roach Hardware Exploration
	Part 5 – Event Detection
	Part 6 – Better Event Detection
	Part 7 – Finite State Machine (FSM)
	Part 8 – Hierarchical State Machine (HSM)