Module 21

Robot Challenges

Robot Challenges

Educational Objectives:

The purpose of the robot challenge is to combine previous modules into a system that solves a complex task. A line-following challenge can be attempted after module 13. You can attempt the simple barrel racing challenge after finishing modules 16 and 17. More advanced challenges will require additional modules. Solutions to the labs provide components (hardware and software) for the challenge. In the challenge, you perform system-level design with the components developed in previous modules.

Good to Know: This challenge allows you to perform duties typical of the engineering profession. The first typical engineering duty is expansion or modification. In other words, given a system that works, how might we reuse the solution to solve a similar but slightly different problem? For example, you interfaced 6 switches in lab 10, and now you might wish to add a 7th switch. The duty would then be to rework the 6-switch solution so it now allows 7 switches. The second engineering duty is integration. In other words, given two systems that work, how might we combine the two systems to create a more complex system?

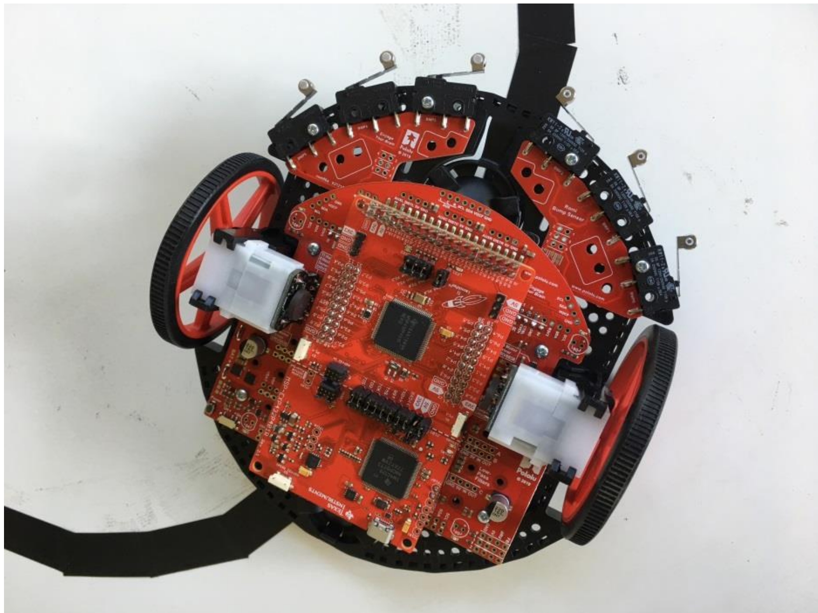

Figure 1. System integration to solve a complex task.

1. Getting Started

1.1. Software Starter Projects

Recall that your low-level drivers are located in the **inc** folder. This approach allows you to reuse code from one lab for the next lab. Together with your solutions to the other labs, look at these two projects, which combine your previous modules into a single system:

Jacki (an empty project up through module 17)
JackiFSM (empty project up through module 13)
Odometry (project using module 16 to move in a square pattern)

If you are attempting a challenge without completing the individual labs beforehand, there are three projects that embed object-code solutions to the low-level drivers. Since the low-level drivers are provided in object code form, this approach will not allow you to learn how it works.

TI-RSLK-MAX (a starter project using just the basic kit)
Competition_IR (a starter project with Lab 15 IR distance sensors)
Competition_BLE (a starter project with BLE)

1.2. Student Resources (Links)

MSP432 Technical Reference Manual (SLAU356)
Meet the MSP432 LaunchPad (SLAU596)
MSP432 LaunchPad User's Guide (SLAU597)
QTRX.pdf, line sensor datasheet
GP2Y0A21YK0F_IR_Distance_Sensor.pdf, datasheet
TI-RSLK MAX user guide
TI-RSLK MAX construction guide document
drv8838.pdf Data sheet for the H-bridge driver

1.3. Reading Materials

Refer to the book readings for each of the modules with which you combine to make your robot.

1.4. Components needed for this lab

Refer to the components needed for each of the modules with which you combine to make your robot. Refer to the **construction guide** for a complete description of how to build the robot.

1.5. Lab equipment needed
Voltmeter
Oscilloscope (one or two channels at least 10 kHz sampling)
Logic Analyzer (4 channels at least 10 kHz sampling.

2. Design Requirements for Basic Challenges

Challenge: Feel free to adapt/combine these ideas into a problem that the integrated robotic system can solve.

2.1. Design Challenge 1: Line following

If your robot has a line sensor and the bump sensors, you could create a line follower that races along a line, and uses the walls to detect when it has strayed far from the line, see Figures 2 and 3. Knowing which bump sensors were triggered tells you the angle it hit the wall. Knowing the angle allows you to back up, turn around and head back to the center of the room, searching for the line again.

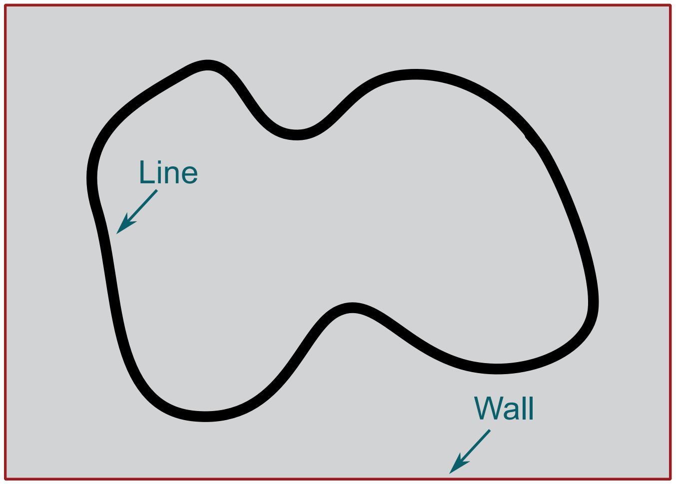

Figure 2. Create a robot explorer that follows a line.

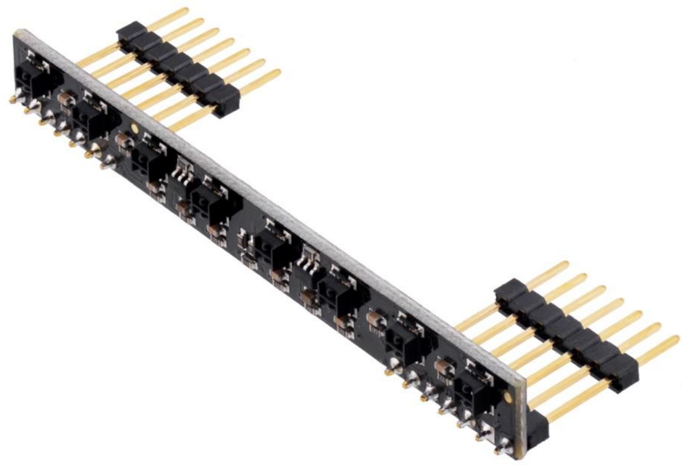

Figure 3. Make sure you use all sensors on the line sensor to create a robot explorer that follows a line.

To solve this challenge, the minimum set of modules you need are:

- *Module 6: GPIO to interface the line sensor*
- *Module 7: FSM as an appropriate approach for line following*
- Module 10: Use SysTick periodic interrupts for the line sensor
- Modules 12+13: Motors and PWM for the robot

Module 14, which is optional, could be used to interface the bump sensors, causing the robot to stop on a collision.

2.2. Design Challenge 2: Line-following to search for treasure

If your robot has a line sensor and the bump sensors, you could create a maze solver that searches for treasure, see Figure 4. Like challenge 1, it uses the walls to detect when it has strayed far from the line. This option will require a high-level maze solving strategy. You will need a mechanism to determine when the treasure has been reached. Possibilities for detecting the treasure include:

- Detecting a special pattern, like 0101010;
- Making the treasure taller than the wall and placing some switches at a height higher than the wall so there switches get triggered only at the treasure.

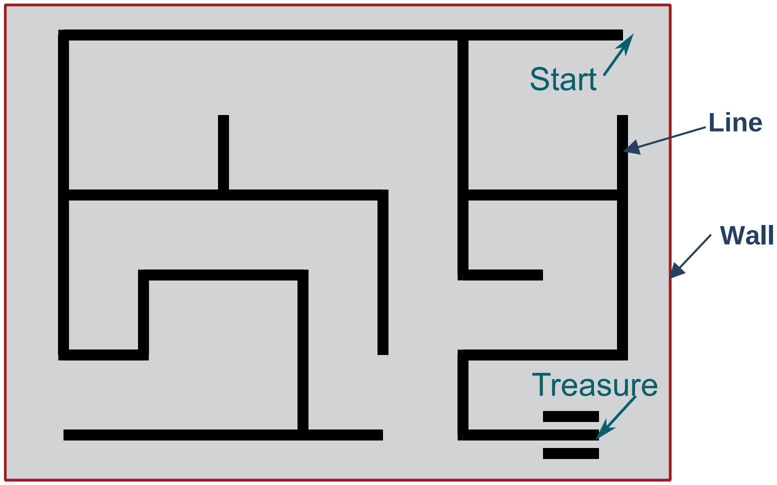

Figure 4. Create a robot explorer that finds its way out of a maze, using just the line sensors and bump sensors.

To solve this challenge, the minimum set of modules you need are:

- *Module 6: GPIO to interface the line sensor*
- *Module 7: FSM as an appropriate approach for line following*
- Module 10: Use SysTick periodic interrupts for the line sensor
- Modules 12+13: Motors and PWM for the robot
- Module 16: Read tachometer data, implement odometry

You can use odometry attempt to move the robot to rotate the robot ±90 degrees. Module 14, which is optional, could be used to interface the bump sensors, causing the robot to stop on a collision.

2.3. Design Challenge 3: Bump and run exploration to find treasure

If your robot does not have a line sensor, you could create a maze solver that searches for treasure using just the bump sensors for guidance, see Figure 5. This robot must bump into the wall as it feels its way around the course. The walls are at a height that can be sensed by some of the bumper switches. The treasure is taller than the walls, and some switches are placed at this higher location, allowing the robot to distinguish between wall and treasure. When the robot has found the treasure, it should stop and flash its LEDs to signify success.

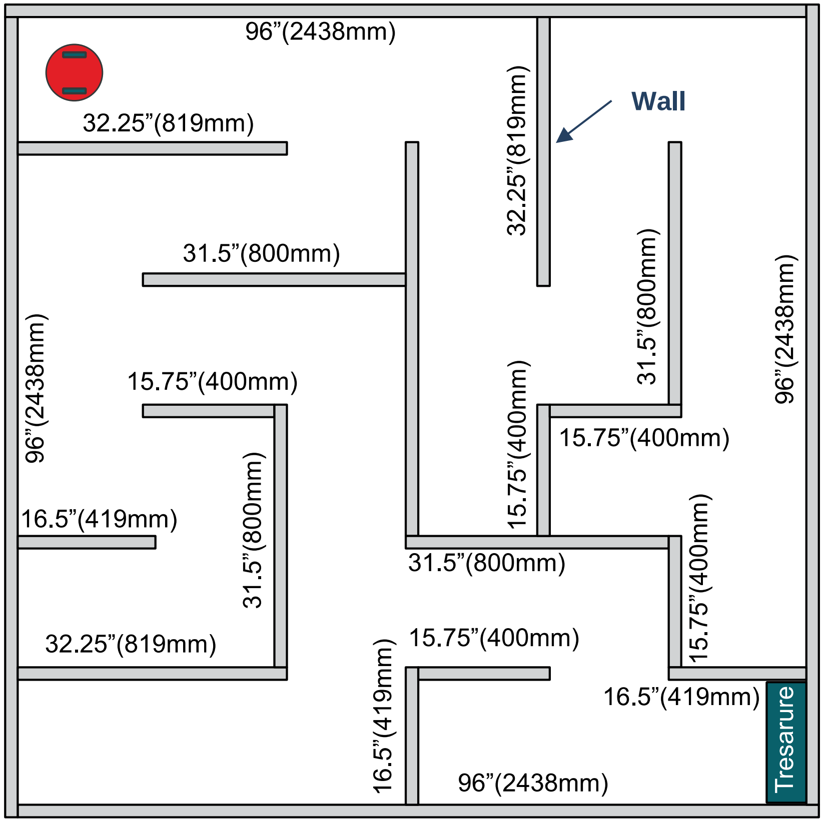

Figure 5. Create a robot explorer that finds a treasure within the maze. The treasure is taller than the walls.

You can use odometry attempt to move the robot in a prescribed distance and direction. If you hit a wall you can use which switch was triggered to reset the position and direction maintained by the odometry.

To solve this challenge, the minimum set of modules you need are:

- *Module 7: FSM as an appropriate approach for searching*
- Modules 12+13: Motors and PWM for the robot
- Module 16: Read tachometer data, implement odometry

Module 14, which is optional, could be used to interface the bump sensors, causing the robot to stop on a collision.

2.4. Design Challenge 4: Barrel racing

If your robot has the tachometer, you could create a robot that races around a barrel as fast as possible. This robot uses the tachometer to implement odometry for guidance. You could also use the tachometer to deploy speed control. Since the errors accumulate with odometry, it is best to attempt only simple tasks, like racing around a barrel, see Figure 6. The field has a starting line and a barrel. You are allowed to know the size of the barrel and how far the barrel is from the starting line. You are allowed to position your robot anywhere in the starting area, located to the left of the starting line. The line sensor can be used to know when robot has crossed the line a second time, and the race is complete. The arena is surrounded by a fenced wall that will trigger a bump switch if your robot gets lost. Your robot should not bump into the wall, but if it does, it is allowed to back up turn and continue. When the robot has travelled around the barrel and crossed back across the line, it should stop and flash its LEDs to signify success. The robot with the fastest time around the barrel and back across the line wins the competition.

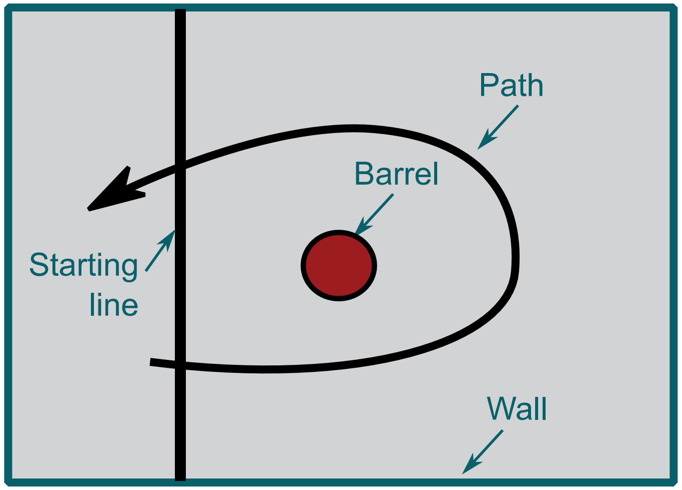

Figure 6. Create a robot explorer that races around the barrel.

To solve this challenge, the minimum set of modules you need are:

- *Module 6: GPIO to interface the line sensor*
- Module 10: Use SysTick periodic interrupts for the line sensor
- Modules 12+13: Motors and PWM for the robot
- Module 16: Read tachometer data, implement odometry

Module 14, which is optional, could be used to interface the bump sensors, causing the robot to stop on a collision. Module 17, is also optional, could be used to create a control system to implement speed control.

Robot Challenges

3. Design Requirements for Advanced Challenges

3.1. Design Challenge 5: Maze exploration

If your robot has distance sensors, you could create a maze solver that searches for treasure using just the distance sensors for guidance. The maze structure shown in Figure 7 could be used. This robot should not bump into the wall. Rather, it uses the IR distance sensors to avoid the walls.

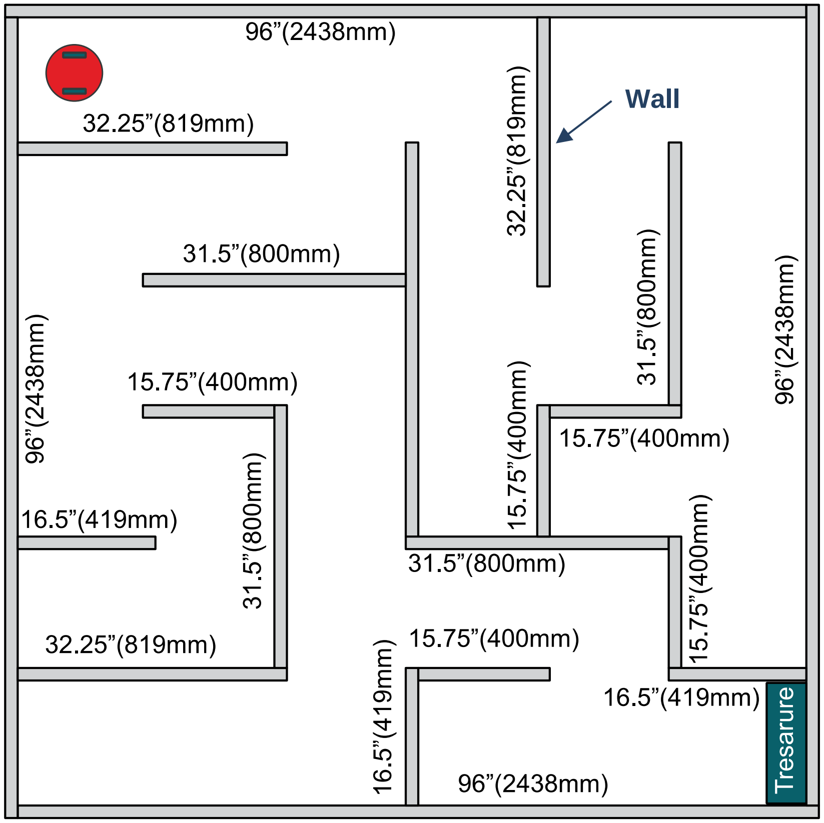

Figure 7. Create a robot explorer that finds a treasure within a maze. The treasure is taller than the walls (same as Figure 5).

The walls are at a height that can be sensed by some of the bumper switches. The treasure is taller than the walls, and other switches are placed at this higher location, allowing the robot to distinguish between wall and treasure. When the robot has found the treasure, it should stop and flash its LEDs to signify success.

To solve this challenge, the minimum set of modules you need are:

- *Module 4: Pattern recognition to help guess which way to go*
- *Module 7: FSM as an appropriate approach high-level design*
- Modules 12+13: Motors and PWM for the robot
- Module 14: Edge triggered interrupts for the collision sensors
- *Module 15: IR distance sensors used to sense the walls.*
- *Modules 19 or 20: Wireless communication (optional)*

Other options for measuring the distance to the wall include ultrasonic sensors (e.g., **ping)))** and **HC-SR04**) and optical sensors (e.g., **OPT3101**).

If you performed either Lab 19 with BLE or Lab 20 with Wi-Fi you could use the wireless communication to add functionality to the challenge.

3.2. Design Challenge 6: Maze exploration

If your robot has both the IR distance sensors and the line sensor, you could create a maze solver that searches for treasure using both sensors for guidance. An example maze structure is shown in Figure 8.

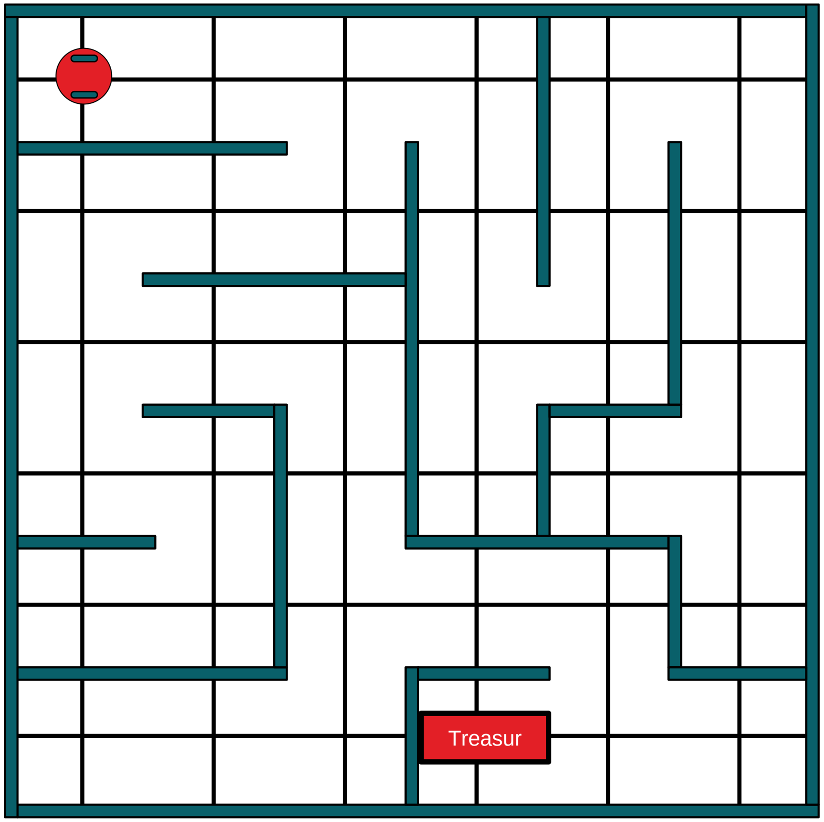

Figure 8. Create a robot explorer that finds a treasure within a maze. The treasure is taller than the walls.

This robot should not bump into the wall. Rather, it uses the line sensors and IR sensors to avoid the walls. The walls are at a height that can be sensed by some of the bumper switches. The treasure is higher than the walls, and some switches are placed at this higher location, allowing the robot to distinguish between wall and treasure. When the robot has found the treasure, it should stop and flash its LEDs to signify success.

With the tachometer and odometry, this robot will be able to turn 90, 180, 270 degrees and move forward a known distance.

To solve this challenge, the minimum set of modules you need are:

- *Module 4: Pattern recognition to help guess which way to go*
- *Modules 6+10: GPIO to interface the line sensor*
- *Module 7: FSM as an appropriate approach for line following*
- Modules 12+13: Motors and PWM for the robot
- Module 14: Edge triggered interrupts for the collision sensors
- *Module 15: IR distance sensors used to sense the walls.*
- *Modules 16+17: Tachometer and control system (optional)*
- *Modules 19 or 20: Wireless communication (optional)*

Other options for measuring the distance to the wall include ultrasonic sensors (e.g., **ping)))** and **HC-SR04**) and optical sensors (e.g., **OPT3101**).

If you performed either Lab 19 with BLE or Lab 20 with Wi-Fi you could use the wireless communication to add functionality to the challenge.

3.3. Design Challenge 7: Autonomous racing

If your robot has the IR sensors but not the line sensor, you could create an explorer robot using just the IR distance sensors for guidance. The goal is to travel around the world as quickly as possible, see Figures 9 and 10. The robot uses the IR sensors to avoid the walls and the other robots. The walls are at a height that can be sensed by the bumper switches. You can race one at a time or you can race multiple robots at the same time.

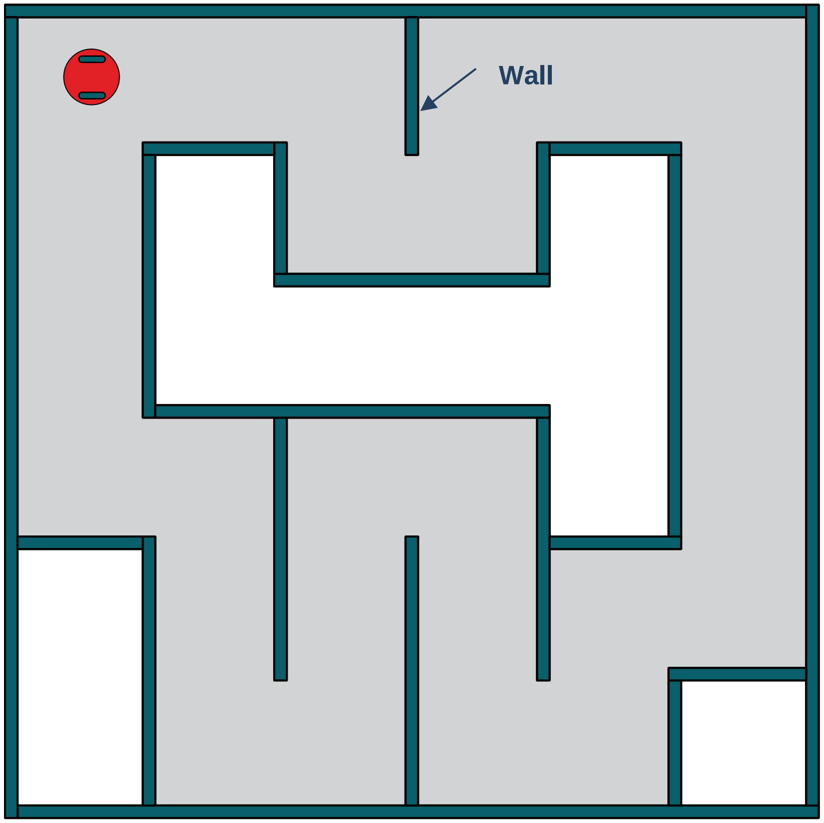

Figure 9. Create a robot explorer that races around a track using bump sensors and IR sensor.

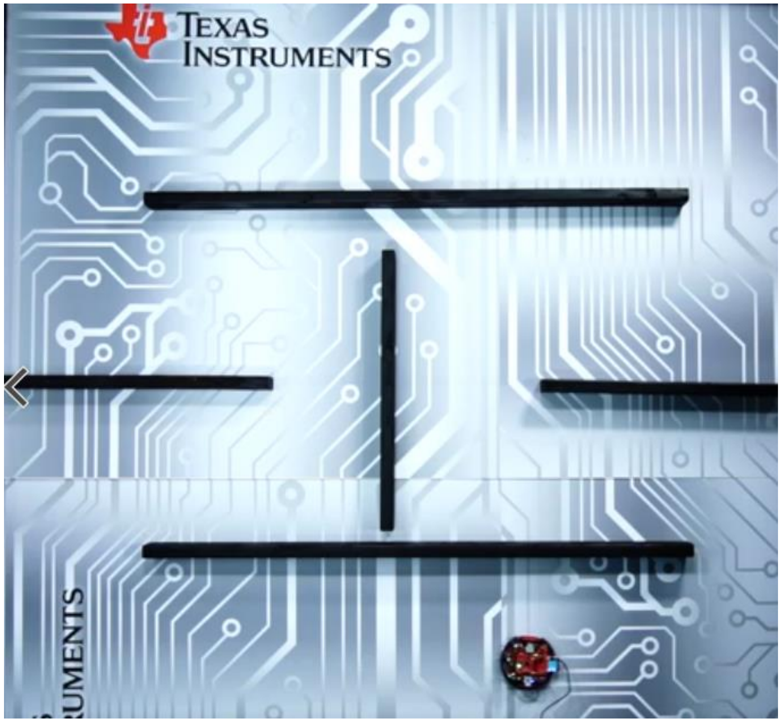

Figure 10. The track is wide enough that the IR distance sensors are monotonic.

To solve this challenge, the minimum set of modules you need are:

- *Module 4: Pattern recognition to help guess which way to go*
- *Module 7: FSM as an appropriate approach high-level design*
- Modules 12+13: Motors and PWM for the robot
- Module 14: Edge triggered interrupts for the collision sensors
- *Module 15: IR distance sensors used to sense the walls.*
- *Modules 16+17: Tachometer and control system (optional)*
- *Modules 19 or 20: Wireless communication (optional)*

Other options for measuring the distance to the wall include ultrasonic sensors (e.g., **ping)))** and **HC-SR04**) and optical sensors (e.g., **OPT3101**).

If you performed either Lab 19 with BLE or Lab 20 with Wi-Fi you could use the wireless communication to add functionality to the challenge.

3.4. Design Challenge 8: Autonomous racing with sensor integration

If your robot has the IR sensors and the line sensor, you could create an explorer robot using both the line sensor and the IR distance sensors for guidance. The difficulty will be to integrate data from both types of sensors. The goal is to travel around the world as quickly as possible, see Figures 11 and 12. The robot uses the IR sensors to avoid the walls and the other robots. It could use the line sensor to orient itself relative to the walls. The walls are at a height that can be sensed by the bumper switches. Because of the width of the robot compared to the size of the track, this challenge was meant to race one robot at the same time.

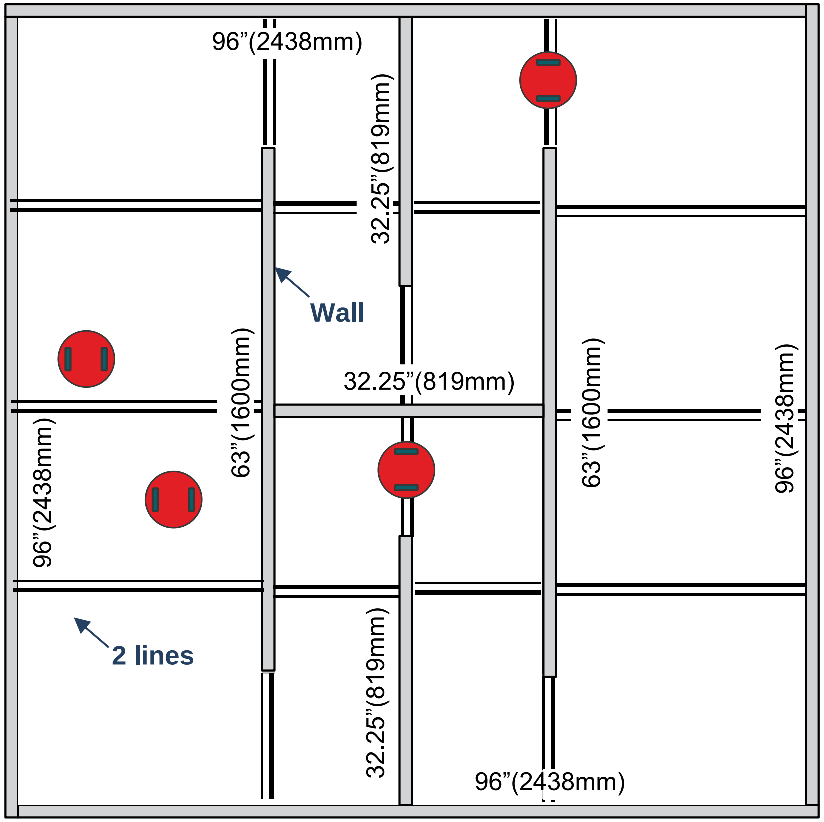

Figure 11. Create a robot explorer that races around a track with bump sensors, line sensors, and IR sensors. Multiple robots can race at the same time. The lines are perpendicular to walls. Each line marking is has two lines (thick and thin) so robot can measure angle and direction.

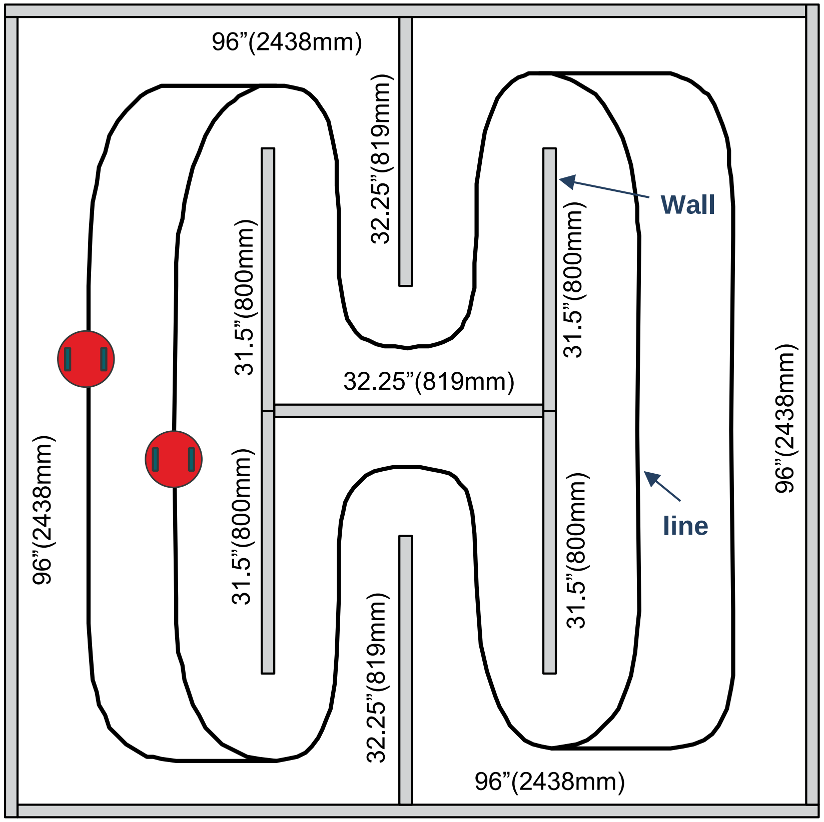

Figure 12. Create a robot explorer that races around a track with bump sensors and line sensors.

To solve this challenge, the minimum set of modules you need are:

- *Module 4: Pattern recognition to help guess which way to go*
- *Modules 6+10: GPIO to interface the line sensor*
- Modules 12+13: Motors and PWM for the robot
- Module 14: Edge triggered interrupts for the collision sensors
- *Module 15: IR distance sensors used to sense the walls.*
- *Modules 16+17: Tachometer and control system*
- *Modules 19 or 20: Wireless communication (optional)*

Other options for measuring the distance to the wall include ultrasonic sensors (e.g., **ping)))** and **HC-SR04**) and optical sensors (e.g., **OPT3101**).

3.5. Design Challenge 9: Synchronized dancing

If you performed Lab 15 with the audio output to a speaker and an audio input with a microphone, you can implement synchronized dancing, see Figure 13. It will be important to design a communication protocol with low latency (time from command sent to time command is recognized). *Designate the robot with the speaker to be the master or conductor. The other robots will interface microphones and sample the sounds, classifying and responding to commands.*

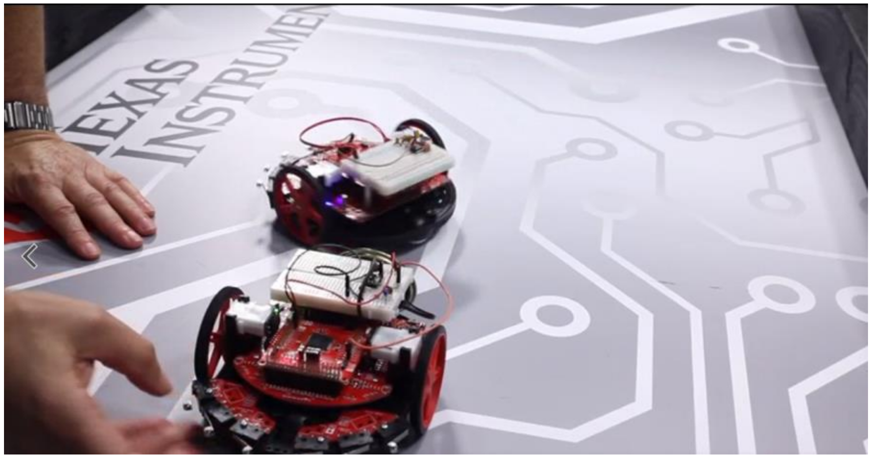

Figure 13. The robots with the microphones will dance to the sounds generated in the robot with the speaker.

To solve this challenge, the minimum set of modules you need are:

- Modules 12+13: Motors and PWM for the robot
- *Module 15: Speaker output and microphone input.*
- *Modules 16+17: Tachometer and control system (optional)*
- *Modules 19 or 20: Wireless communication (optional)*

Using odometry and a control system would allow the dancer to accept high-level abstract commands from the master while the slave controller performed low-level operations locally. In other words, rather than accept commands like {go, stop, turn-right, turn-left, and back-up}, the high-level commands could be {forward-10-cm, left-90-degrees, right-90-degrees, and back-10-cm).

If you performed either Lab 19 with BLE or Lab 20 with Wi-Fi you could use the wireless communication to add functionality to the challenge.

4. Experiment set-up

For each module you choose to deploy refer back to that module to configure and hardware and software needed. To build the arena, the walls need to be heavy enough so the robot does not push it over.

Construction approach 1: One very flexible approach to building the arena is to use individual 4 by 4 pieces of wood, see Figure 14. Wood this size is heavy enough to be placed on the floor without fastening the pieces together.

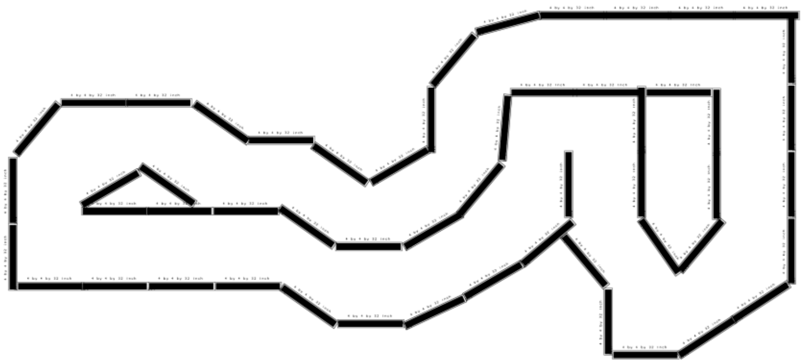

Figure 14. The Formula 1 race track at Austin Texas can be built with 54 pieces of 4 by 4 wood.

Construction approach 2: Another approach to building the arena is to use individual 2 by 4 pieces of wood cut to specification. The mazes shown in Figures 2 through 11 can be built with pieces of 2 by 4. Wood this size is not heavy enough to be placed on the floor without fastening the pieces together. So these pieces will need to be screwed or hammered together, which is why the designs all have 90 degree angles.

The robot is 6.43 inches (163 mm) round. The arena is 8 foot by 8 foot. You can think of the space as 36 individual rooms, each room is 15.75 inches square (400mm). Walls are 1.5 in wide (so effective room size or door clearance is 14.25 inches, 362 mm).

Pieces need to construct the mazes shown in Figure 5, 7, and 8 are listed in Table 1.

count	in	mm
4	96	2438.4
4	15.75	400.05
3	16.5	419.1
4	31.5	800.1
3	32.25	819.15
1	47.25	1200.15

Table 1. Materials needed to build the maze shown in Figures 5,7,8.

Multiple mazes by flipping and rotating (start and end in different corners), as shown in Figure 15.

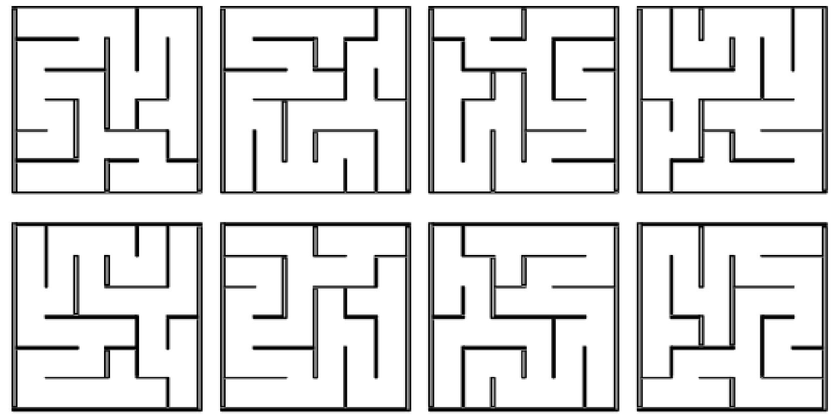

Figure 15. Multiple mazes can be made by rotating and flipping.

Figures 10, 11, and 12 define race tracks. Racing is a fun class activity and really rewards good design. Table 2 shows the pieces needed to construct these race tracks.

count	in	mm
4	96	2438.4
2	63	1600
3	32.25	820

Table 2. Materials needed to build the maze shown in Figures 10-12.

5. System Development Plan

5.1. Top-down design

The entire curriculum was based on the bottom-up approach, starting with simple components. After you learned how a component operates, it was abstracted, creating a set of functions you need to use it. In the challenge, however, we will take a top-down approach. There are five phases of the project:

1. Analysis (requirements, specifications, constraints)
2. High-level design (strategies, data flow graph, algorithms, abstractions)
3. Low-level design (call graph, header files, data structures, flow charts, how will it be tested?)
4. Implementation (modularity, concurrent development)
5. Testing (bottom-up testing, control and observability)

Strategy. You will begin with developing an overall strategy. An important decision is which sensors to use and how you plan to use them. You will need a plan to balance speed with accuracy.

The finite state machine is one approach to consider for implementing line following. The FSM from Lab 6 had only 2 inputs, see Figure 14. Your solution had more states, but it maintained the 2 input/ 2 output structure. One option is to distill the line center data to create the 2 inputs as envisioned when you solved Lab 6. Furthermore you can use the FSM output values to set the duty cycle for each motor.

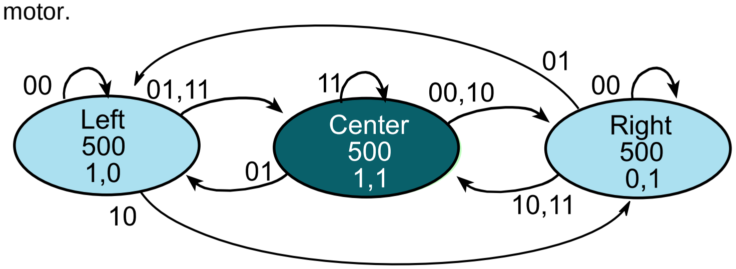

Figure 14. Moore FSM state graph to implement line following. The time in each state is shown in 1ms units.

If you review your Lab 6 results you will find the line sensor output is not a continuous variable, but rather can take on only a finite number of values. Furthermore if you distill the line sensor data into eight distinct classifications, you can use these eight discrete input possibilities to drive a line following robot similar to Lab 6. Each state in the FSM has 8 next-state arrows.

- Lost
- Way off to left
- Off to left
- Little bit left
- Centered
- Little bit right
- Off to right
- Way off to the right

A Fuzzy Logic controller could also be used to follow the line.

Module 17 introduced a number of control algorithms you could use to move in a straight line or follow along a side wall.

5.2. System integration

Once an overall plan is developed you need to integrate components from the other labs to create a system that performs an overall task. Consider how priority is assigned. Consider how the different software threads are integrated into one system. For example, you could define four threads

- Periodic SysTick interrupts to measure the line sensor
- Periodic Timer A1 interrupts to run the high-level strategy
- Edge triggered interrupts for collisions
- Main program for debugging and low priority tasks

Recall that the PWM outputs to the motor do not require software action to operate. Once configured the Timer_A hardware automatically produces the PWM outputs. Software needs to execute only when the direction or duty cycle are to be changed.

Next, you need a mechanism to pass data between threads. Semaphores, mailboxes, and FIFO queues are appropriate for this project. For example, the ISR might just stop the motors and set the Collision mailbox. Subsequently, the Timer A1 periodic thread can handle the collision, backing and turning as needed.

5.3. Testing

Effective debugging involves *control* and *observability*. Four strategies for controlling what your robot does (without going through the edit, compile, download, debug cycle) are

- Connect the robot via a long USB cable and run the debugger
- Use switches on the robot to select various modes/strategies

- Connect the robot via a long USB cable and implement an interpreter via the UART on the MSP432 and a terminal emulation program running on the PC. This way you can send commands and observe responses (Lab 18).
- Interface the CC2650 and interact with the robot over BLE (Lab 19)
- Interface the CC3100 and log data onto the cloud (Lab 20)

Four strategies for observing internal parameters within your robot as is attempts the challenge are

- Connect the robot via a long USB cable and run the debugger
- Output strategic parameters to the LCD/OLED (Lab 11)
- Connect the robot via a long USB cable and output debugging information to the UART. Observe the information using a terminal emulation program running on the PC (Lab 18).
- Interface the CC2650 and interact with the robot over BLE (Lab 19)
- Interface the CC3100 and log data onto the cloud (Lab 20)

5.4. Team management

"A team is a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they are mutually accountable."

1. (Katzenbach, J.R. & Smith, D.K. (1993). The Wisdom of Teams: Creating the High-performance Organization. Boston: Harvard Business School.)
2. Decker, Philip, J. (1996) "Characteristics of an Effective Team," http://www.cl.uh.edu/bpa/hadm/HADM_5731/ppt_presentations/29teams
3. Breslow, L. (1998). Teaching Teamwork Skills, Part 2. Teach Talk, X, 5. http://web.mit.edu/tll/published/teamwork2.htm. 13 May 2003.
4. Building Blocks for Teams, (Website). Penn State University, http://tlt.its.psu.edu/suggestions/teams/student/index.html

Respect. Team effectiveness requires mutual respect. It is expected that team members will disagree. Managing conflict will be easier from a position of mutual respect. When arguing with your teammates, please consider strongly the possibility that you might be the one who is wrong.

Communication. Team effectiveness requires effective and constant communication. Create a channel (googledoc, slack, signal etc.) where thoughts and interactions can be recorded. Listen attentively and respect your teammates. Ask lots of questions. Give constructive feedback. Present your ideas forcefully, but keep an open mind. Restate the original idea to be sure it's understood. Critique the idea, not the person. Be courteous. Be aware of body language and tone. Meetings don't need to be a death march, use humor effectively. Laugh with someone, do not laugh at someone.

Leadership. A leader is responsible for 1) calling meetings; 2) finding a mutually agreeable time and place to meet; 3) setting a meeting agenda; 4) facilitating the meeting; 5) monitoring progress against the plan; and 6) identifying problem areas that need action. The leader is not "the boss". The team needs to agree on decisions and directions. Compromise is essential.

Effective Meetings. Before the meeting, name someone to be the facilitator, create an agenda and send it to all team members. Set a time limit for the meeting. During the meeting, if issues emerge that are not on the agenda, the facilitator should: ask the team if this should be discussed now, or table the issues for the end of the meeting. During the meeting, keep a list of decisions and actions items, keep to the time commitment, create an agenda for next meeting and agree on time and place. After the meeting, send out a brief summary of a list of action items and those responsible for those actions.

Brain storming. Select someone to be the recorder. Invite everyone to give their ideas and input. Write down all ideas without criticism or discussion. Avoid being judgmental of others' ideas. Try to look at all sides of an idea. Listen attentively and treat your teammates' opinions with respect. Try to encourage the widest range of new ideas. Everyone should participate. Don't stop the idea session too soon. After complete list is generated, return for discussion/analysis. Carefully select the best approaches or ideas from the list. Try to remove your ego from the discussions. Don't take the rejection of your ideas personally.

Code repository. Use a code repository, with version control, so multiple members of the team can work simultaneously.

Fail fast. Identify the component of the project that has the most risk (least likely to work), and determine quickly if it will be feasible. Always develop a plan B when proposing a strategy that may be risky.

Be realistic, be simple. Many teams fail because they attempt a strategy that is too complex. Many teams get frustrated over the size of the project, and over thoughts that they are working alone. A simple strategy often yields acceptable results when time is constrained.

Be the team that is having the most fun!

Effective team checklist

- Define a common goal for the project.
- List tasks to be completed.
- Assign responsibility for all tasks.
- Develop a timeline and stick to it.
- Develop and post a Gantt chart for the plan

- Document key decisions and actions from all team meetings.
- Send reminders when deadlines approach.
- Send confirmation when tasks are completed.
- Collectively review the project output for quality

IMPORTANT NOTICE AND DISCLAIMER

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265