ME 348: Advanced Mechatronics

Laboratory 2: State Machine Design

Submit one report in Canvas as a team. Include the names of all teammates. Be sure to submit your completed worksheet (a scan) and any required plots in a single file (.pdf format), with your code attached at the end of the pdf. Please also upload any arduino or python code as well. The due date is given on the Canvas assignment. This lab will span two lab periods.

Overview
The purpose of this lab is twofold. The first goal is to understand how to use serial communication to get the Raspberry Pi and A-Start to exchange information. The second goal is to understand how to develop and build finite state machines through a design problem, specifically to write and test code for how your robot should respond to different obstacles, as detected by the bump switches on the robot.

Components Required: You will need a Romi chassis with motors and bump sensors, the Raspberry Pi and A-star controller.

I. Lab Preparation Activities

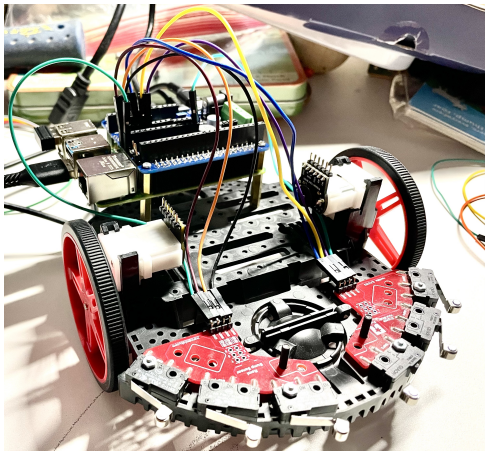
A. Obtain the template files for this lab from the Git Repository: `https://github.com/AnnMFey/ME348_AdvMech_Lab2`. Clone this repository to your Pi and navigate to it using your terminal so you can easily commit and revert any changes you make.

B. Install the AStar Library. Instructions for doing so can be found in the link: `https://pololu.github.io/a-star-32u4-arduino-library/`

C. Install numpy and serial (a.k.a. pySerial) python libraries. Use commands `sudo pip install pyserial` and `sudo pip install numpy` .

D. The Arduino and Rpi communities rely on open-source documentation and tutorials in order to pass along information. One of the most shared Arduino tutorials is user Robin2's Serial Input Basics tutorial, linked here: `https://forum.arduino.cc/t/serial-input-basics-updated/382007`. Spend a few minutes reading at least the first two sections and be ready to reference this throughout the lab.

E. Finally, you will need to mount your bump sensors and the servo motors on the Romi chassis. If you don't have the right screws to mount the bump sensor boards, they can be found on the parts cart in lab. You will also need to obtain 20 F/F jumper wires to hook up the motors and sensors. The Romi board is not designed for the M2.5 screws used by the Raspberry Pi but it is possible to use some of the holes and standoffs (provided) to mount the board sufficiently for this lab. See the instructor's example - yours could look similar to the image below. For now, just physically mount the boards and motors. We will wire them up later.



II. Serial Communication

A. Examine the repository now cloned on your Rpi. You will see four folders labeled step 1 through step 4. Inside each folder is an .ino file to be uploaded to the Astar board, and a .py file to be run through vscode on your Pi. Upload the .ino file in the step 1 folder to the Astar and open the .py file in VScode. Run the .py file, you should see <Arduino is ready> being printed to the vscode terminal.

B. On line 7 the serial communication port is initialized between the Rpi and the Astar by calling an instance of the Serial class. Hover your mouse over the word Serial (capital S) and you should see a list of potential variables that the serial class could take in. As an exercise in reading documentation, go to the Pyserial webpage and describe below what the purpose of one of the potential input variables are (Don't pick one of the two used for this lab, which are port and baud rate)

C. Move on to the Step 2 folder, upload the .ino file to your Astar and run the python file from your Pi. These pieces of code send a message back and forth between the Rpi and the Astar. The function to really pay attention to here is the recWithStartEndMarkers() function. Using the Serial Input basics tutorial as a reference, explain some reasons why we much prefer to use a function like this to one of the built in Arduino functions that would allow the Astar to read data send across the serial port. Also explain how these pieces of code prevents the input and output buffers of the Astar and Rpi from becoming clogged with data.

Consider spending some time messing around with the variable Send2String to see if you can break the program; add <'s, take them away, add characters outside the <>'s, etc., is the communication between the Astar and Rpi robust? Could you make it better in your future robot code?

D. Once you are satisfied you understand what is happening in step 2, move on to the step three folder. Upload the .ino file to the Astar, and open the .py file with VScode. Also use a barrel jack connector attached to a 9V battery or wall outlet to supply power to the motor drivers on the Astar. Use the Astar user's guide to figure out how to connect the motors to the motor drivers (`https://www.pololu.com/docs/0J66`), or come reference the finished robots at the instructors' table.

E. Safety Note: In the next step we will make the motors spin. It is highly recommended that you prop your robot up so it does not drive away. Stopping the .py program will not stop the motors. If you need to suddenly stop the motors, it is recommended you unplug the barrel connector.

F. Once everything is connected, take a look at the code. The main new thing going on here is that once a string is sent from the Rpi to the Astar, the Astar parses that data from a string into 2 ints. In the box below, please explain what the strtok and atoi functions do, as well as why we have to go through the trouble of converting between data types.

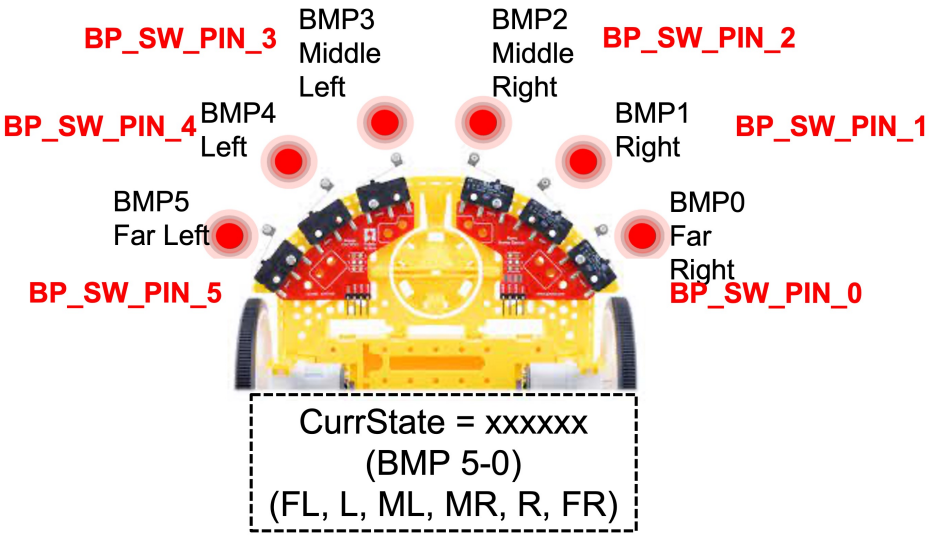
Also take some time to look through the Astar 32U4 robot controller's Arduino library (`https://www.pololu.com/docs/0J66/5`) Identify the functions setM1Speed and setM2Speed and make the motors spin as fast as they can. Get used to looking through and looking for libraries, there is no need to reinvent the wheel if you can find libraries you can use!

G. Once you feel comfortable with your understanding up step 3, move on to step 4.

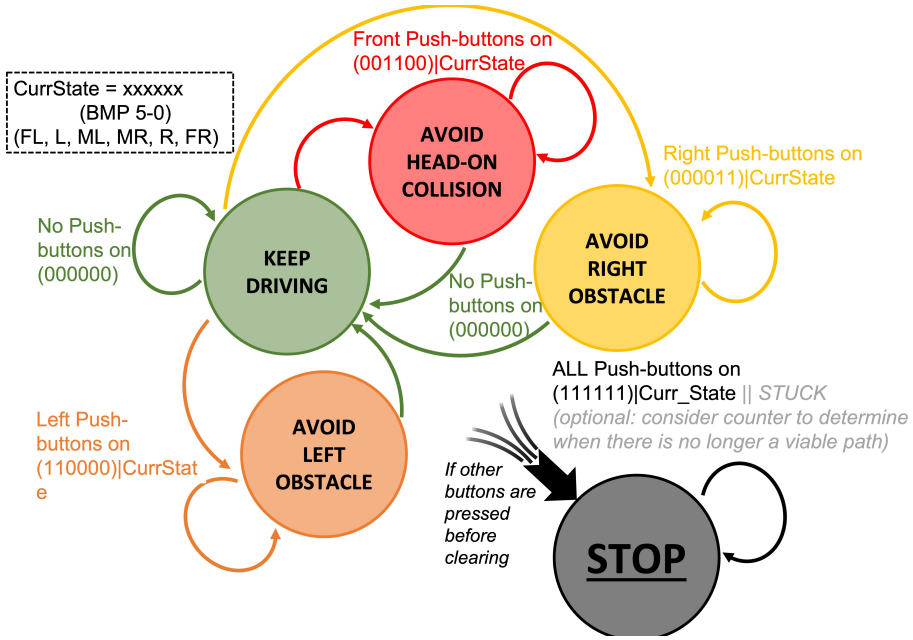
Use the Astar user's manual to figure out what pins you need to hook the bump sensors up to. Feel free to reassign the pins to somewhere else. The template code uses 0,1,4,5,7,8 arbitrarily. Take some time to look at the code, you should notice that now we are now parsing data on both the Astar AND Rpi side. Also take note of this code using two additional software tools available in python including the ability to write custom functions and use try/except cases (Arduino typically disables try-catch blocks to conserve computational resources). We wrote the sendString() function to address a low level bug that was causing the bits in the Astar's input buffer to be overwritten, and we used the try/except case to identify packet loss. Think about why checking for packet loss is good practice in general. Take some time to look over these pieces of code so you prepare yourself to use them to write a state machine.

III. Problem Statement and FSM Design

A. First we will create a preliminary version of an obstacle avoidance strategy for the robot. The robot has 6 bump sensors labeled as BMP0 (far right) - BMP5 (far left). Select some more meaningful names for the bump sensor variables, as well as the pin numbers that will be associated with them. For example, the pin assignments could be BP_SW_PIN_0 - BP_SW_PIN_5 and the bump sensor variables could be FR for Far Right, R for Right, MR for Middle Right, ML for Middle Left, L for Left, and FL for Far Left. This is the naming convention used in the images below but your team is free to chose a different convention. To keep the lab simple, we will initially use adjacent pairs of bump sensors to define the states for obstacle avoidance (e.g., if BMP0 and/or BMP1 is triggered, we need to avoid an obstacle to the right, similarly for BMP3 and BMP4 and a potential head-on collision). The image below shows the location of the bump sensors and suggested variable names.



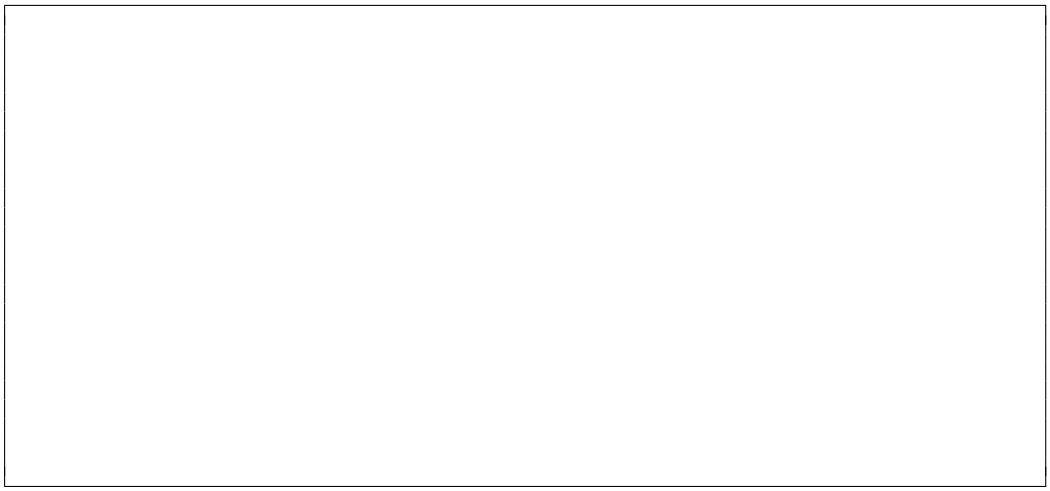
B. Below is a preliminary state machine diagram for our obstacle avoidance strategy. Transitions between states occur when different sets of push-buttons are triggered (i.e., either of the adjacent pairs or sensors OR both).



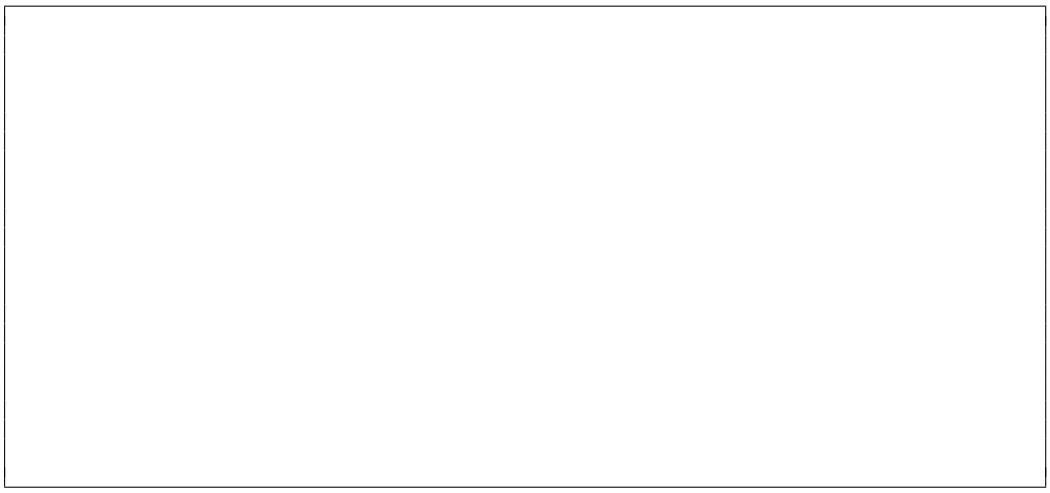
C. Write some *basic* psuedocode below for what your robot should do in each of the following states. We will try to follow the example in `https://github.com/bminch/PIE/blob/main/FSM1.ino` in which each state has an associated software function to perform some necessary task, as well as check for state transitions, and set or clear important variables to enable smooth transition. Functions in the AStar library code which may be useful for the motors specifically can be found here: `https://pololu.github.io/a-star-32u4-arduino-library/class_a_star32_u4_motors.html`. Your pseudocode does not need to be fully written as code yet, just show some thought behind the design of each function (i.e., what motors to control, timing considerations, etc).

i. KEEP DRIVING

ii. AVOID HEAD ON COLLISION



iii. AVOID RIGHT OBSTACLE



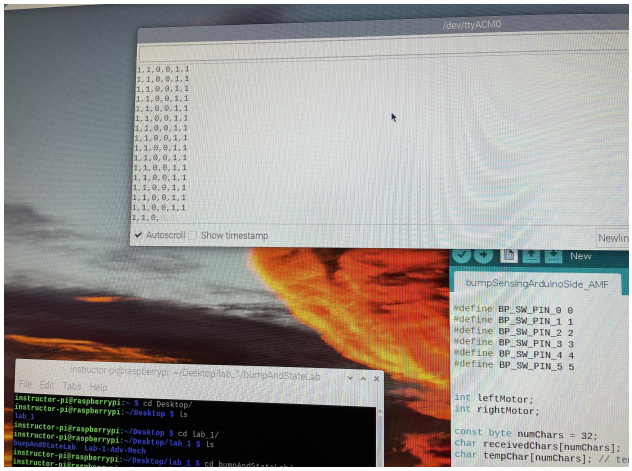
iv. AVOID LEFT OBSTACLE

v. STOP

IV. FSM Implementation

A. Now we will program the FSM on your robot.

B. As a first step, let's make sure the push buttons work. The template code provided in github will send serial characters to the raspberry pi based on what button in pressed. We can easily test the buttons in the Arduino environment first by copying the digital read lines and serial print lines found in the SendBumpData() function and pasting them temporarily at the beginning of the loop function. Make sure all buttons work and then rename variables to help make more sense out of your code. Use define statements to associate pin numbers you will use to a variable to make it easier to not lose track of pin numbers later in the code (note, define statements do not use semicolons). We used pins 0-5 for this but you should spend time studying the pin diagrams for the A-Star as these may need to change as you add more functionality to your robot: `https://www.pololu.com/docs/0J66/3.9`. You should see something like this on the serial monitor (make sure your baud rate is correct). In this example, the middle two buttons are pressed. Remove those lines of code and re-upload to the A-Star controller.



C. Next, develop your basic state machine for obstacle avoidance using a switch case statement in your python code (this is the same as the Step 4 folder). This is an open ended activity so there may be many ways of doing this. Be sure to look at the FSM programming style found in the link below as a guide `https://github.com/bminch/PIE/blob/main/FSM1.ino`. This code will likely just be a first pass at a solution, try not to get too bogged down in perfecting the algorithm as there will be time for this as we build up the rest of your robot.

D. Tip: Try to first code in an e-stop that sets the motor speeds to zero when a certain combination of bump sensors or buttons are pressed. This will make resetting and iterating your code a lot easier.

E. Include your code with the lab report submission.

V. FSM Testing

A. With any design process, developing a testing plan is critical. Develop and describe a systematic testing plan to evaluate your push-button FSM and report your results. Did all your transitions occur correctly? Are there any surprises or forgotten states? Did the motors respond as intended? Show the TA.

B. (Post-Lab Question) What sort of protections might you need add into your system to deal with multiple, simultaneous user inputs? Also, does your system have any states that depend on prior states? Is your code fully non-blocking? Describe how you might improve your code in the future.

VI. Final Deliverable (to continue next week).

A. The servo motors have encoders that can be used to measure wheel rotation. The information about the encoders can be found on this product page: `https://www.pololu.com/product/3542`. Figure out how to connect your encoders to the A-Star controller and use the Arduino Encoder library to measure the speed of your wheels (`https://www.arduino.cc/reference/en/libraries/encoder/`). This will be very helpful later. In a future lab, we'll develop our own PID controllers for these motors as well as an external motor.