MEMORANDUM

To: Charlie Refvem, Professor, Department of Mechanical Engineering, Cal Poly SLO
crefvem@calpoly.edu

From: Matthew Wimberley — wimberle@calpoly.edu
David Reo — dreo@calpoly.edu

Date: June 14, 2023
RE: **Term Project Deliverable #2: Final Memo Submission**

Mechanical

The mechanical hardware of our design was much more familiar to us than the electrical/PCB design portion of the term project. We started with a rough CAD model of our robot frame but quickly deviated as PCB sizing and sensor placement commanded the individual facets of our frame. Using the 3D printers available to us in Mustang 60, we built our entire frame from PLA, using fasteners and adhesive bonds to integrate pieces with each other. We went through several iterations of each component including our base, ball retriever, ramp, and motor container. The ball retriever required the most iterations as the torque generated from our ball retriever sometimes sent the ping pong ball outside the bounds of our frame. To counteract this, we printed the ramp and base board again with guard rails to keep the ping pong ball from flying outside our frame, as shown in the figure below.

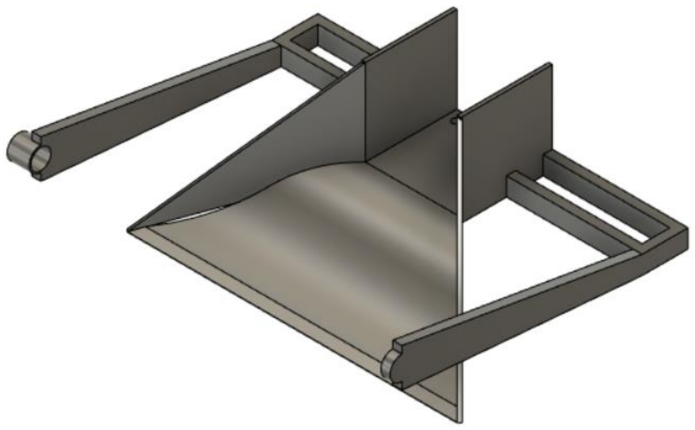


Figure 1. Ramp with Guard Rails

Also, the ball retriever had a tough time capturing the ping pong balls with our initial iteration, so we switched to a toothed gate design which was far more reliable at retrieving the ping pong balls and transporting it up our ramp.

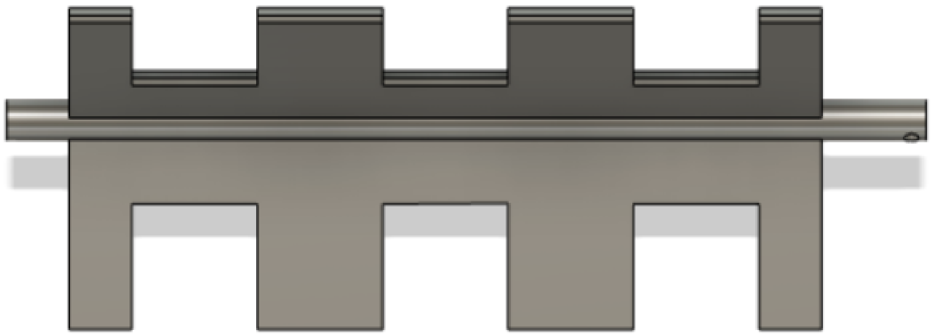
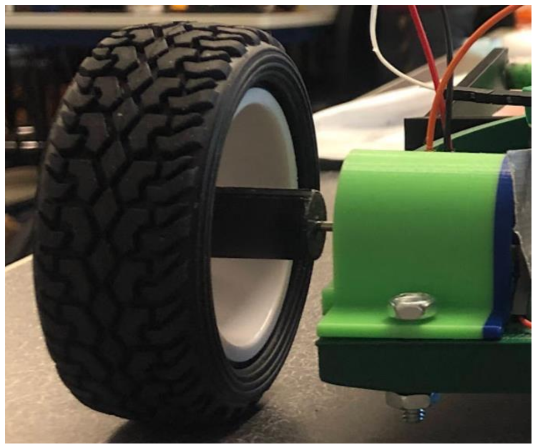


Figure 2. Toothed Gate Ball Retriever

We were able to capture a few ping pong balls, as depicted in our video attachment, but getting the motor to spin with instantaneous torque proved challenging due to the high friction between PLA pieces. To combat this, we would give an initially greater duty cycle to get the ball retriever spinning then immediately switch to an appropriate duty cycle given the light weight of our components. This proved to work in spurts, but after some 5 seconds of spinning the ball retriever would dislocate from the collar supports on the ramp. In a future iteration we would make the collars about 10 millimeters longer to account for lateral translation of the ball retriever relative to the frame of the robot.

One PLA mechanical component that worked very well was our motor strap. It took a few revisions, but in the end, we contrived a support that could fasten the oscillating motor to the base, spin the wheel without sputtering, and conveniently make the motor accessible. That is what the vertical bars in the figure below illustrate. Due to the oscillatory nature of motors, our wheels would translate laterally (perpendicular to the direction of motion) relative to the base, so we added these supports, and it improved the overall performance of the wheel-motor system.



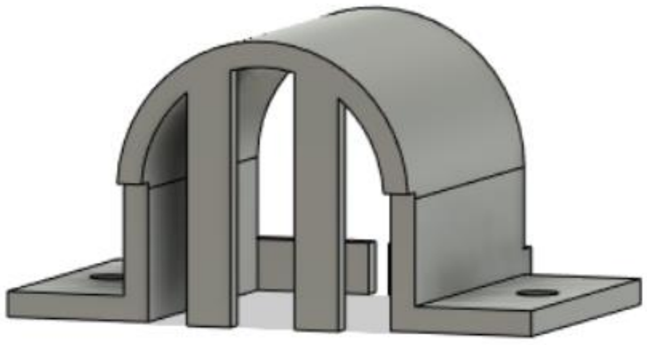


Figure 3. Motor Strap Side & Isometric View

In conjunction with our motor strap was our motor coupler, which was custom fitted to our motor and hexagonal shaped wheel hub. This provided instantaneous torque to our wheel, as the hexagon shape inserted into the wheel provided 0 slippage due to the tight tolerancing foresight for our motors. Unfortunately, we did not pick motors with a D-shaped shaft, so we filed one side of our motor shafts down to give better grip for a set screw to latch onto.

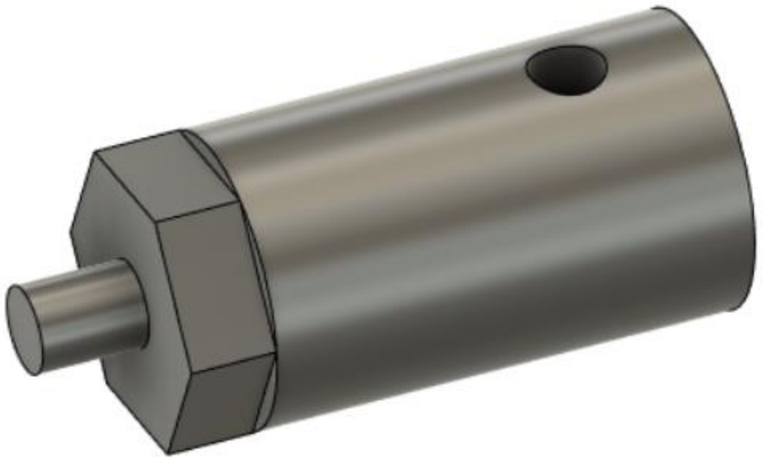


Figure 4. Motor Coupler

In the end of our mechanical design, we arrived at some impasses that were not thoroughly designed beforehand. For example, the ultrasonic sensor located at the back of our robot had to be pivoted slightly so that the line sensors can directly be on either side of the center axis. This made our line following logic easier as discussed in the physical modeling section.

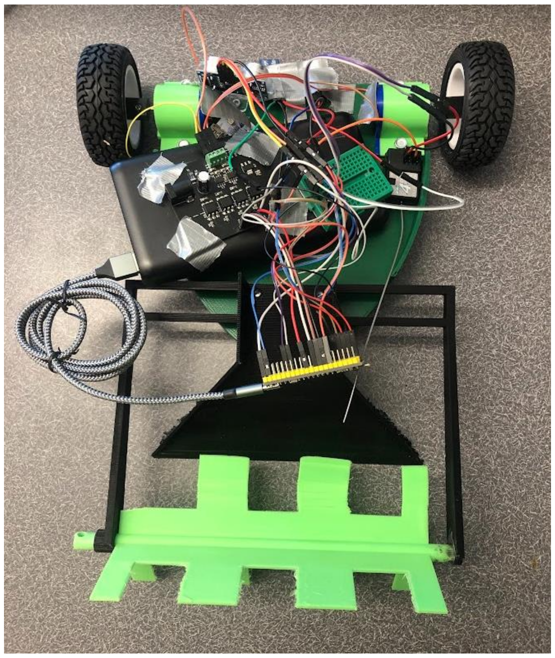


Figure 5. Robot End Design with Black Pill, Sensors All Located at Top of Image

Electrical

Sensors and Actuators

Concerning our electronic hardware, the motors and sensors were also hit or miss. The DC brushless motors we used mimic those in mechatronic courses, so we were able to use the same motor driver for our personal DC motors on our robot as the motors ran at the same frequency and had equivalent auto reload and period values. As for the line sensors, we initially picked an analog line sensor, unaware of the need for an analog-to-digital Converter, so we borrowed a digital line sensor from another group and attached them to the base of our base board. With this digital setup, however, we could only read the relevant GPIO pin as high or low; there was no gradient for the voltage reading from the line sensor, so we had to use more than one. Accordingly, we borrowed another digital line sensor and placed it at the base of our base board, about 2 centimeters to the side of the first line sensor, with the centerline equidistant between those two sensors, as shown in the figure below.

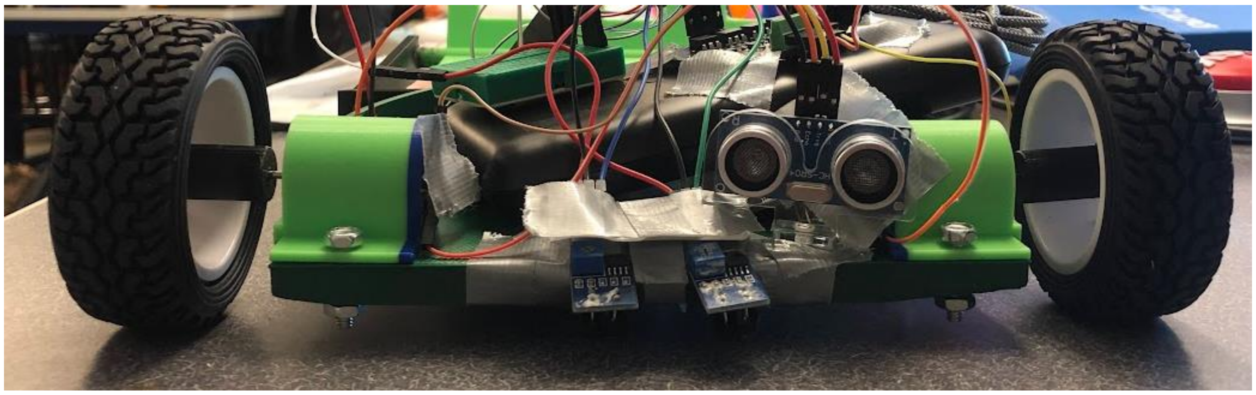


Figure 6. Line Sensors Equidistant About Centerline

We briefly also worked on the implementation of a servo motor to kick out colored balls that were not of our team's color; however, this was within a tight timeframe before the demonstration day, so we were unable to integrate this into our robot. The idea was to have a color sensor be at the forefront of our ramp, so that incorrectly retrieved balls can be checked as soon as they entered our base board, then a 90-degree bar, like an Allen wrench, flicks undesired balls simply off our board, and only balls of our color remain. To move these correctly colored balls to our home goal, another servo motor would kick the balls from our depot station only after our robot found its starting home.

The color sensor by far proved to be the most challenging of all our actuators. Calibrating the sensor was not a consistent measure as the sensor seemed to necessitate different readings to even be remotely usable. Due to the time constraints of our term project, we chose to focus on more tangible achievements like getting the robot motors to react depending on input from the line sensors and ultrasonic sensors.

Speaking of which, the ultrasonic sensor was as easy to work with as our line sensors. Meant to stop if another robot was detected too close to it, the ultrasonic sensor provided another safeguard to protect our robot from potential collisions. Our ultrasonic sensor, the SEN-15569, had multitudes of guides to reference, unlike our color sensor, so this was our quickest actuator to get working properly.

PCB Design and Testing

Our circuit board was designed for a 12V battery and included a 12V to 5V buck regulator to support many of the sensors as well as a 5V to 3.3V linear regulator for the MCU. Like most of the class, we decided to stick with the STM32F411 microcontroller. This allowed us the flexibility to use the Black Pill Development Board if our PCB were to not work for any reason.

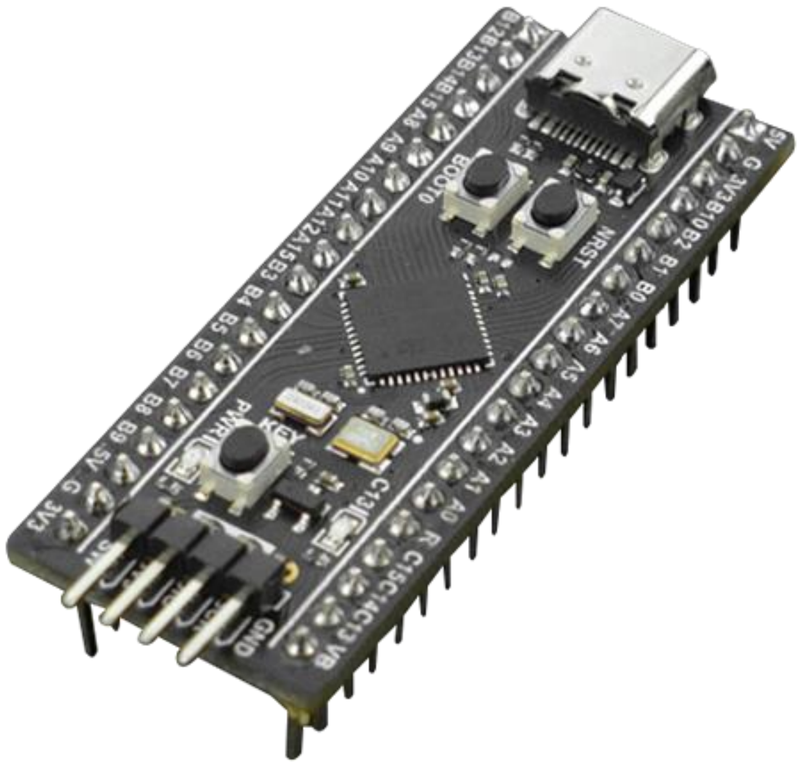


Figure 7. STM32F411 Black Pill Development Board

Other notable elements of the board include an indicator light to show the board is being powered, a 3A fuse to protect the power supply, a MOSFET to allow for reverse polarity (I.e., the board does not go up in smoke from hooking the battery up backwards), a 20MHz crystal, and three motor drivers. In addition, most of the ICs on the board required circuit elements (resistors, capacitors, inductors, diodes, etc) to be placed adjacent to them that could be sized from the related datasheets.

We soldered all the elements onto the board using a stencil with solder paste and a hot plate. In retrospect, a frame would have saved about 2-3 hours of time in applying solder paste to the board through the stencil. There could have been additional time savings from labelling the circuit components according to the label on the PCB (I.e. a 10uF capacitor could be labeled as C3).

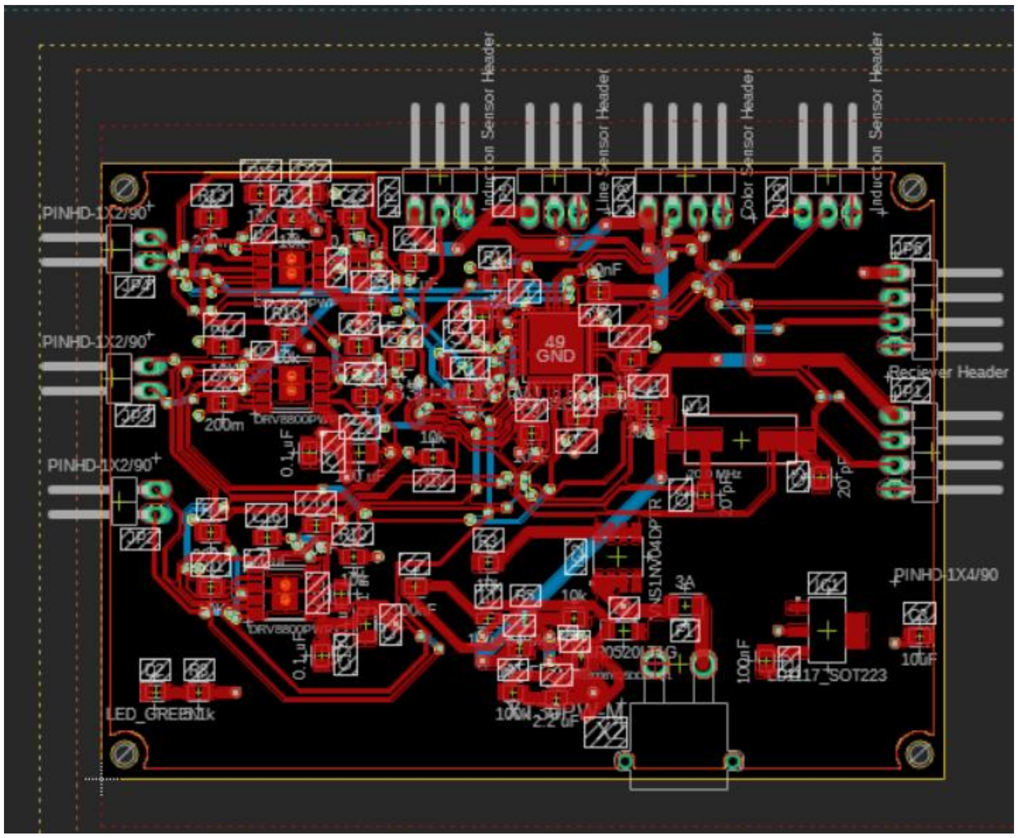


Figure 8. PCB Layout

Testing of the board was an ongoing process that involved a couple interesting MacGyvered techniques. The first phase of testing involved hooking the board up to a 12V power supply with current being limited to 100 mA to prevent the board from being fried. Immediately, it was noticeable that the voltage would not go past 3V, indicating a short on the board. To find the short, continuity testing was performed on several circuit elements. This proved to take quite a lot of time, and considering the tight time budget, we quickly switched to a different technique. This technique involved putting a thin layer of 91% isopropyl alcohol on the board, then once again hooking it up to the power supply. With current being limited as before, the location on the board where the isopropyl would evaporate the quickest would then indicate the location of the short. This proved fruitful, indicating three locations of shorts. The first was a motor driver which was successfully reworked, while the second and third were the buck regulator and linear regulator. The regulators were each removed and then continuity was retested to confirm that there were no other shorts. With both regulators off the board, there was no apparent short anymore. Both were replaced, and continuity was tested once more: the short was back. After several hours of further continuity testing, and a few jumper wires, several of the surrounding components on each regulator were replaced to no avail. At this point, it was determined that there was a possible defect on the board shorting an internal layer. With that hypothesis, the other blank boards were tested but they did not appear to have the same defect. A decision was then made to switch over to the Black Pill considering the tight timeline to get a functional robot. If there were more time to further test, I suspect that one of the components surrounding either regulator was not properly specified, or the wrong component was placed down. This is considering that the short was able to be successfully localized to the regulators. In fact, there was a humming noise that occurred from one of the components surrounding a regulator. There was an attempt to localize the noise by holding up a straw to the tester's ear, but further testing was abandoned in consideration of the deadline.

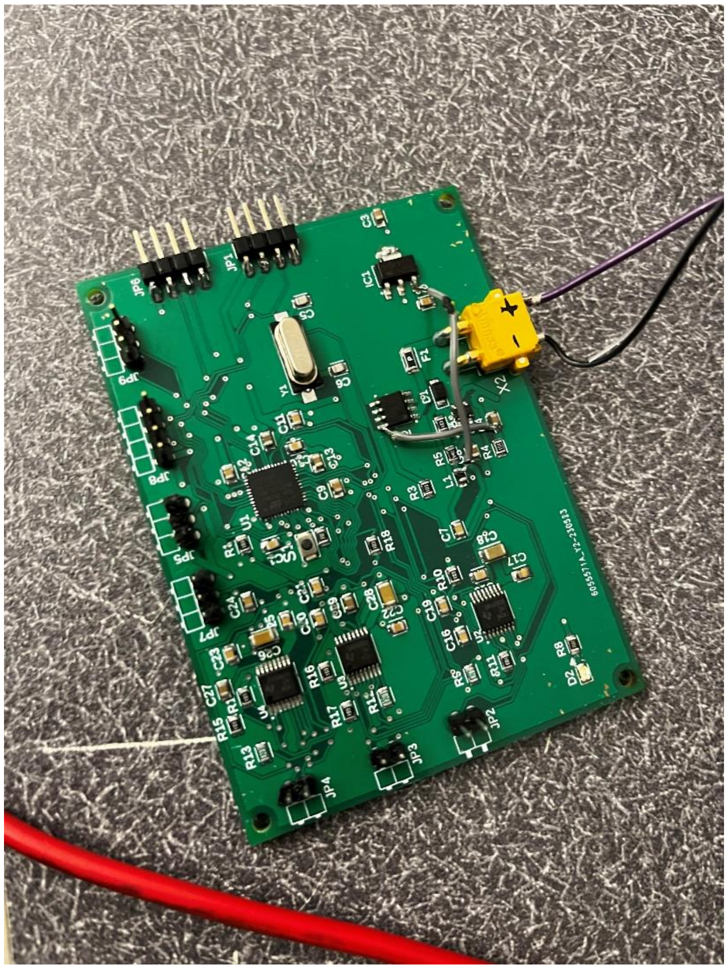
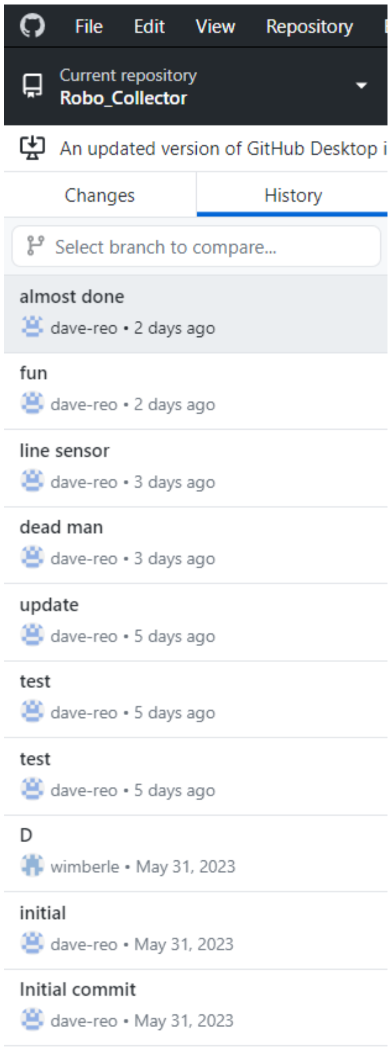


Figure 9. PCB with Electrical Components

Software Implementation

The strategy with our software was to implement the code in the simplest possible manner while leveraging pre-written drivers as much as possible. This strategy is in alignment with the tight deadline and an additional disdain for needlessly complex code. That being said, elements of the code – written entirely in C – that are of interest include a motor driver, setup for the ultrasonic Sensor, the line sensor, and the start/stop ("Deadman's) switch. The commit history can be seen below as well.



almost done
dave-reo • 2 days ago

fun
dave-reo • 2 days ago

line sensor
dave-reo • 3 days ago

dead man
dave-reo • 3 days ago

update
dave-reo • 5 days ago

test
dave-reo • 5 days ago

test
dave-reo • 5 days ago

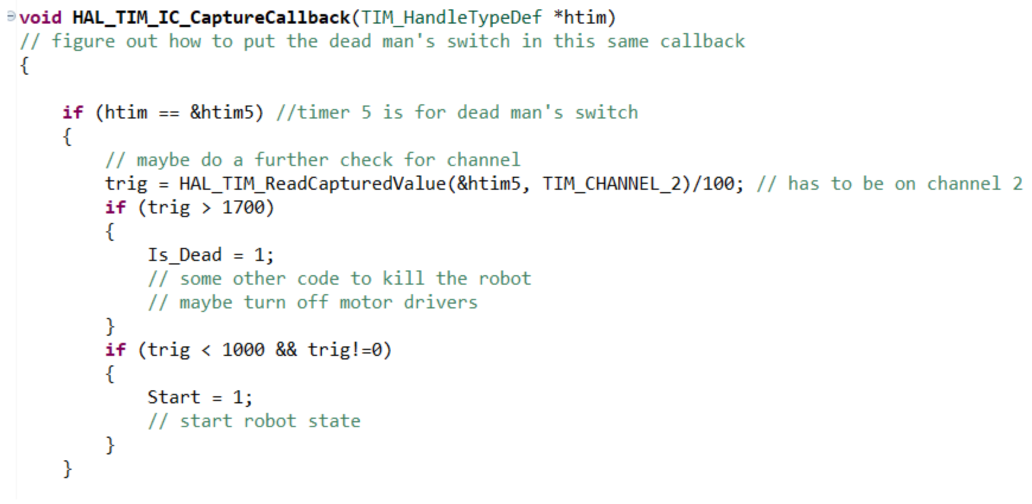
D
wimberle • May 31, 2023

initial
dave-reo • May 31, 2023

Initial commit
dave-reo • May 31, 2023

Figure 10. GitHub Commit History

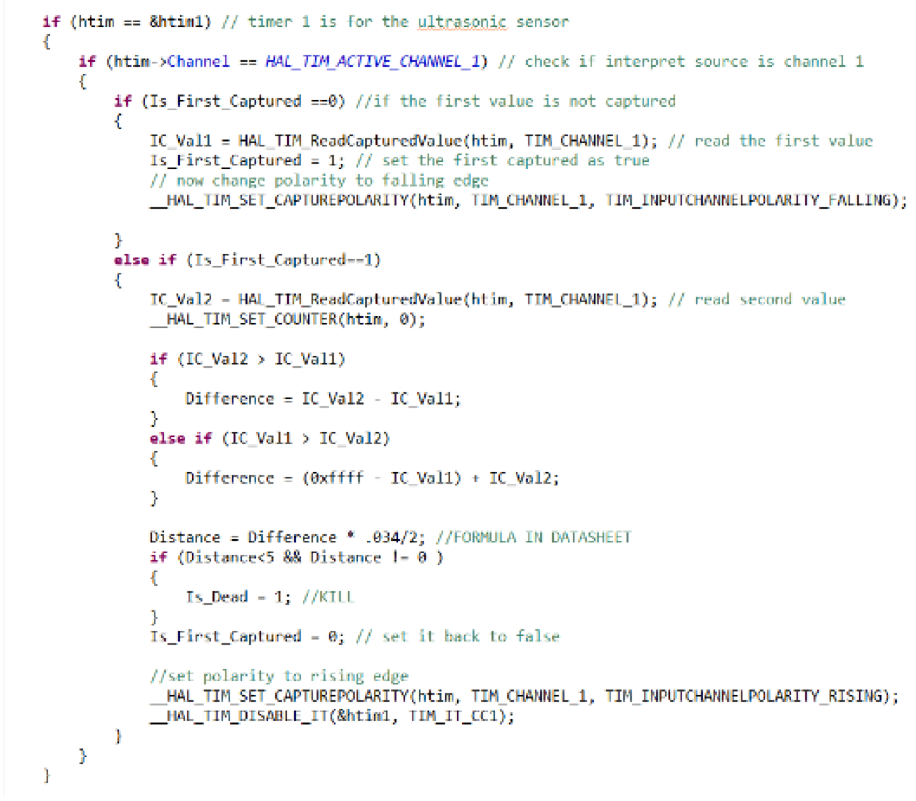
The motor driver was copied from an earlier lab that leveraged pseudo object-oriented programming techniques. The motor driver had an enable and disable function that was critical to the creation of a FSM for the Deadman's switch. There was also a drive function that allowed a duty cycle to be passed into the motor object on a scale from -100 to 100. The Deadman's switch was created by checking which direction a trigger on the transmitter was pulled. Assuming the trigger was pulled down, a flag called Is_Dead would be thrown high, and the motors would be disabled in main by calling the aforementioned disable function. Likewise, another flag called Start would enable the motors when the trigger was to be flicked up.



```c
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
// figure out how to put the dead man's switch in this same callback
{

    if (htim == &htim5) //timer 5 is for dead man's switch
    {
        // maybe do a further check for channel
        trig = HAL_TIM_ReadCapturedValue(&htim5, TIM_CHANNEL_2)/100; // has to be on channel 2
        if (trig > 1700)
        {
            Is_Dead = 1;
            // some other code to kill the robot
            // maybe turn off motor drivers
        }
        if (trig < 1000 && trig!=0)
        {
            Start = 1;
            // start robot state
        }
    }
```

Figure 11. Trigger Capture Callback

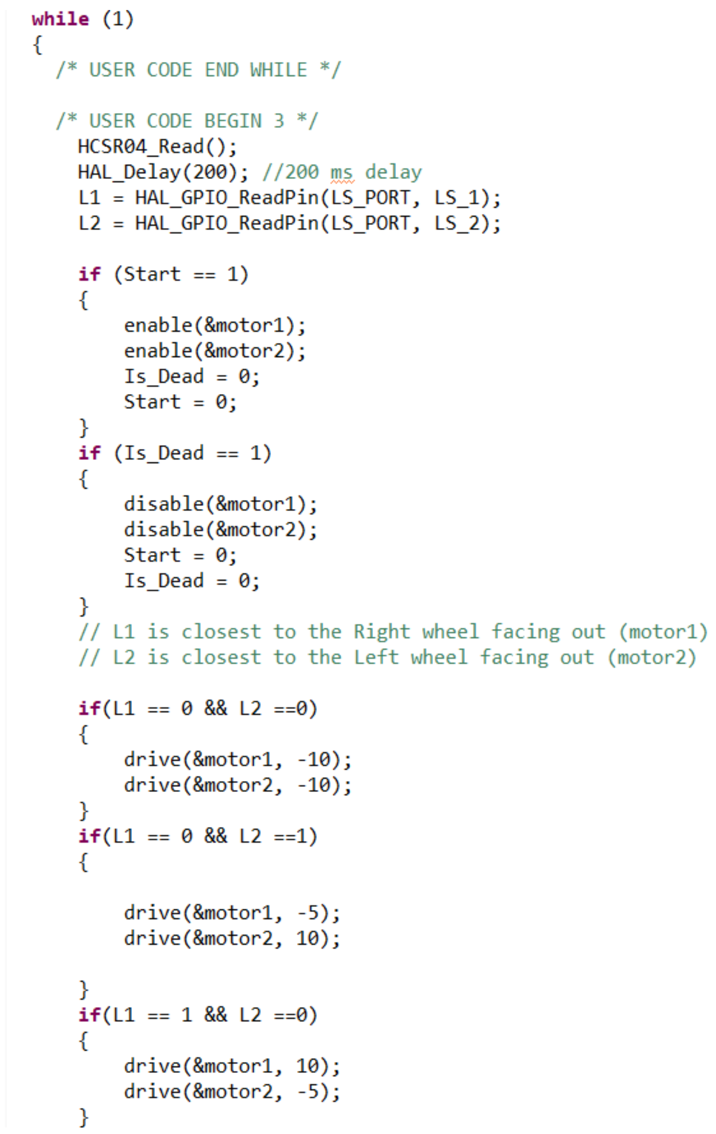
A HAL_TIM_IC_CaptureCallback function was written to check for information from either timer and then implement the expected behavior. Assuming there was not a trigger input, there were also continuous inputs from an ultrasonic sensor that lived on Timer 1. This sensor was adapted from a pre-written driver. One key change, however, was the implementation of the Is_Dead flag going high when the calculated distance met a certain threshold. This would make it so the robot would not run into another robot.



```c
    if (htim == &htim1) // timer 1 is for the ultrasonic sensor
    {
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) // check if interpret source is channel 1
        {
            if (Is_First_Captured ==0) //if the first value is not captured
            {
                IC_Val1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // read the first value
                Is_First_Captured = 1; // set the first captured as true
                // now change polarity to falling edge
                __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_FALLING);

            }
            else if (Is_First_Captured==1)
            {
                IC_Val2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1); // read second value
                __HAL_TIM_SET_COUNTER(htim, 0);

                if (IC_Val2 > IC_Val1)
                {
                    Difference = IC_Val2 - IC_Val1;
                }
                else if (IC_Val1 > IC_Val2)
                {
                    Difference = (0xffff - IC_Val1) + IC_Val2;
                }

                Distance = Difference * .034/2; //FORMULA IN DATASHEET
                if (Distance<5 && Distance != 0 )
                {
                    Is_Dead = 1; //KILL
                }
                Is_First_Captured = 0; // set it back to false

                //set polarity to rising edge
                __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1, TIM_INPUTCHANNELPOLARITY_RISING);
                __HAL_TIM_DISABLE_IT(&htim1, TIM_IT_CC1);
            }
        }
    }
```

Figure 12. Ultrasonic Sensor Capture Callback

Next, the line sensors were entirely implemented in main. We initially had analog sensors, but quickly switched over to a set of digital ones that were left over from a classmate's project. That allowed simple GPIO reading and basic logic to follow based on which sensor was seeing a line. This decision to switch from analog to digital was key to saving time and allowing for more efficient code implementation.



```
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
    HCSR04_Read();
    HAL_Delay(200); //200 ms delay
    L1 = HAL_GPIO_ReadPin(LS_PORT, LS_1);
    L2 = HAL_GPIO_ReadPin(LS_PORT, LS_2);

    if (Start == 1)
    {
        enable(&motor1);
        enable(&motor2);
        Is_Dead = 0;
        Start = 0;
    }
    if (Is_Dead == 1)
    {
        disable(&motor1);
        disable(&motor2);
        Start = 0;
        Is_Dead = 0;
    }
    // L1 is closest to the Right wheel facing out (motor1)
    // L2 is closest to the Left wheel facing out (motor2)

    if(L1 == 0 && L2 ==0)
    {
        drive(&motor1, -10);
        drive(&motor2, -10);
    }
    if(L1 == 0 && L2 ==1)
    {

        drive(&motor1, -5);
        drive(&motor2, 10);

    }
    if(L1 == 1 && L2 ==0)
    {
        drive(&motor1, 10);
        drive(&motor2, -5);
    }
```

Figure 13. Main Loop

Many of the elements of the main loop seen above have been described previously. The notable thing to look at is how neatly written the loop is. There is a continuous reading of the ultrasonic and line sensors with a FSM for either motor enabling or disabling, and there is logic to drive the motors based on how the line is read.

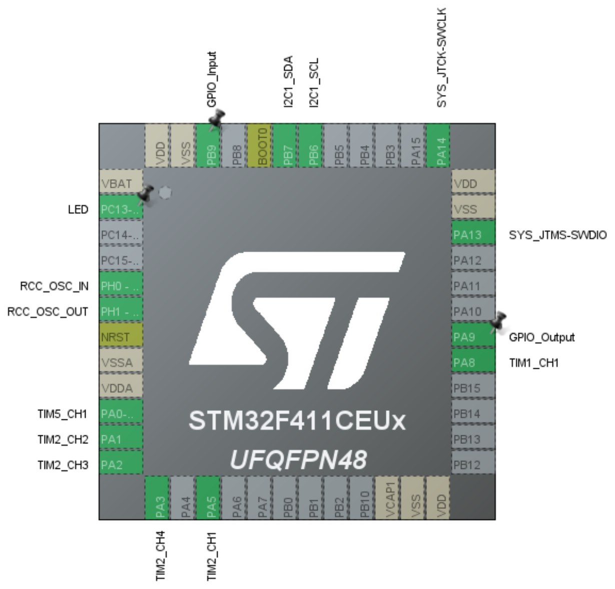


Figure 14. STM32 Pinout View

Finally, the above pinout view describes how the pins on the Black Pill were setup. Timer 5 was used for the transmitter while Timer 2 generated PWM for the motors and Timer 1 was set aside for the ultrasonic sensor. There were also two pins used for I2C on the color sensor, but they were never implemented due to time constraints.

Physical Modeling

To maneuver our robot, we relied on our ultrasonic and line sensors. We never got to test a fully integrated system with all sensors working simultaneously, but individually all the sensors could send signals and the motors were able to react accordingly, minus the color sensor as mentioned before. Due to the immense time constraints this project imposed, we were never able to retrieve a ball then subsequently analyze it for if the ball had to be removed by one of our servo motors. Given another week, our next step would have been to solidify the color sensor and continue work on post-processing sensor data. The current line sensor pair we had set up proved sensible in small scenarios where the line sensor followed a line although in the circular arena, the pair of line sensors proved to not be enough to contain our robot within the bounds of the arena. An array of at least four line sensors all pointed down at the base of our board would likely have sufficed our need to stay within the arena boundaries. Our current logic commanded the robot to turn in the direction that exactly one line sensor was reading low. When both sensors were reading the same value, both high or both low, the robot kept moving straight as in both driving wheels received equivalent duty cycles. Because exactly one of the two line sensors reading low meant the robot was tilted in the yaw axis, we simply had to switch the direction of the duty cycle to the motor that was reading high to keep the robot moving in a counterclockwise motion. We tested that the inverse was true, too, as commanding a negative duty cycle to the line sensor side reading low would keep the robot moving in a clockwise direction. This is further depicted in the video appended to this memorandum. The modification of having four digital line sensors would give us greater “bandwidth” for our line sensors, so we could tell how much we needed to increase the duty cycle in one wheel so as not to overcompensate and overshoot following the line.

Attachments

Full Code Listing

This showcases every detail of the code while further explanation can be found in the above software section.

Ball Retriever

In this three second clip, the ball retriever mechanism can be seen at the front as it is actuated by a third motor, using a custom made block fastening the ball bearing roller to our motor container. The color sensor as well as other electronics are not present, but by itself the retriever is functional, able to transport ping pong balls up the ramp onto our base board.

Ultrasonic Sensor + Kill Switch

The kill switch has been upgraded from Lab 4, this time in conjunction with the ultrasonic sensor where power to the motors can be cut off by the kill switch on the transmitter or an object being too close to our ultrasonic sensor.

Line Sensor Logic

If both line sensors read the same value, the robot will continue moving straight (both motors receive equivalent duty cycles). If exactly one sensor sees a line, our current logic dictates that the line sensor reading high (sees the line) will command a negative duty cycle to the wheel on the same side.