Kendall Gordinier: 662020130 & Jamie Draper: 662024490
Final Project
Embedded Control
April 20, 2022

Project Introduction and Description:

For this project we were tasked with designing and implementing a system embedded in a microcontroller robotic car that would allow the car to successfully traverse a variable maze of obstacles. Programming predefined paths were not allowed as a solution to this project and would only only show completion of a fixed maze, and would not show the ability to adapt to a change in structure of the maze from trial to trial.

Proposed and Final Solutions:

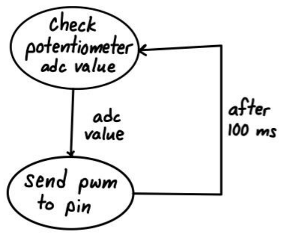
In order to solve this maze, we developed a system of remote control to change the direction of the car moving at a constant speed through an infrared signal, using 2 microcontrollers: one for the car and one for the remote. The controller consisted of a rotary potentiometer, where the direction in which the potentiometer was pointing represented the direction in which the car would turn. When the infrared signal was out of range, the car would continue turning in the last received direction from the remote. With this solution, we were able to easily steer the car through the maze without any trouble.

High-Level Design:

The main function of this system can be broken down into 3 main parts: sending infrared signal, receiving infrared signal, and setting wheel speed based on that infrared signal.

Sending infrared signal:

The infrared pulse-width-modulated signal is generated based on the ADC value of a potentiometer. Every 100ms the ADC value corresponding to potentiometer voltage is read and converted to a PWM signal which is sent to an infrared LED.

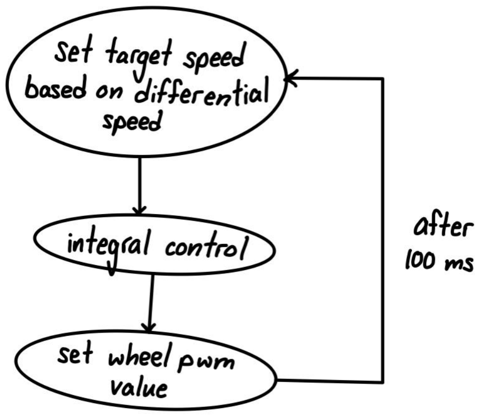


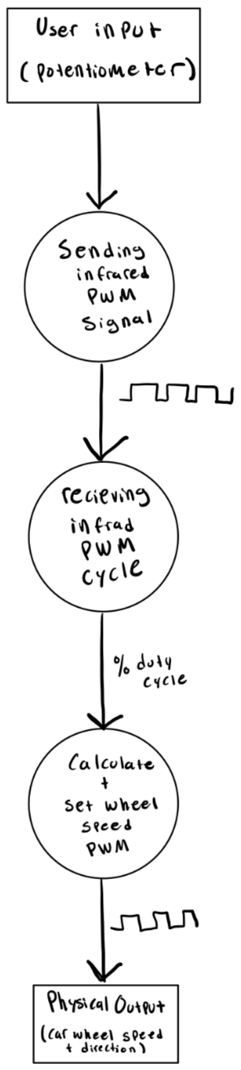
Receiving Infrared signal:

We receive a PWM signal from the remote through a phototransistor. We must convert this signal into a differential speed which can then be used to control the direction of the car. The input that the microcontroller received was the PWM signal from the infrared LED. From this signal a differential speed for each wheel is calculated

Setting wheel speed based on infrared signal:

Every 100 ms we will take the current differential speed from the input PWM and set a target speed for each wheel based on that speed. Then we will implement an integral control scheme for accuracy of speed. We will finally take the PWM output of the integral control and send it to the wheels to modify the speed.

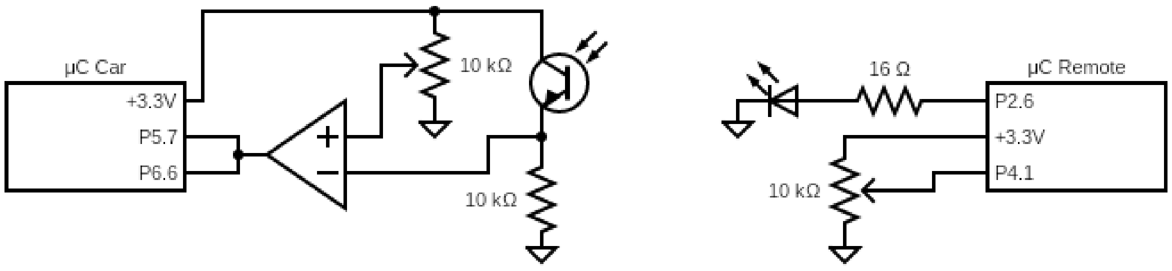




User input (potentiometer)
Sending infrared PWM signal
recieving infrad PWM cycle
% duty cycle
Calculate + set wheel speed PWM
Physical Output (car wheel speed + direction)

These 3 main parts work together to convert a user input from the potentiometer to a physical change in the speed of the car. Data is passed through different mediums across each part to eventually represent this physical change. We start with a sensory input which is transmitted through the air in the form of infrared light. This is then transferred through internal processes of the microcontroller to resemble a PWM signal, which is converted through the electrical and mechanical processes of the DC motor to represent a physical wheel speed that represents the specific user input.

Low-Level Design:



Sending infrared signal:
A pulse-width-modulated infrared signal was sent from the remote microcontroller(right on circuit diagram) based on a rotary potentiometer representing direction. The potentiometer voltage was sent to the ADC and converted to a 14 bit number. This number was converted to a compare value from 50 to 799 out of 800 timer counts representing the PWM duty cycle of the infrared signal based on the equation:

$$\texttt{uint16\_t}\ x = (ADC\, result)\frac{749}{16384} + 50$$

This way we made sure every PWM signal could be read as an input being triggered by rising and falling edges(a value of 0 or 800 would have no rising and falling edges and therefore could not easily read as a PWM input). This way, a potentiometer turned left would give a voltage of 0 to pin 4.1, resulting in x = 50, or turned all the way right would give a voltage of 3.3V, resulting in x = 799. This compare value was then used to set a timer compare value, which generated a PWM signal on pin 2.6 to the LED, producing corresponding pulses of infrared light.

Some example values:

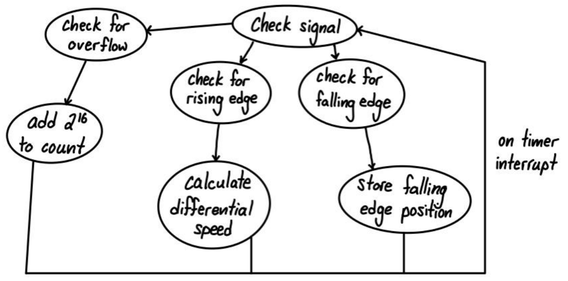
Pot Direction	Voltage	ADC Value	x	Duty cycle
Left	0	0	50	6.26%
Straight	1.65	8192	424	53%
Right	3.3	16384	799	99.9%

Receiving Infrared signal:

To receive the infrared signal on the car's microcontroller, we used an infrared phototransistor to produce high and low values to an input pin, with respect to the high and low values of the infrared signal. At far distances, the amplitude of this signal was too small to represent a difference between logic 0 and logic 1, so we used a comparator and potentiometer(see circuit diagram) to set a threshold voltage where any voltage above that would become 3.3V(1) and anything below would become 0V(see verification section for oscilloscope output).

Setting wheel speed based on infrared signal:

To measure the signal coming from the comparator, we sent it to two separate pins: 5.7 and 6.6, on which we used the capture registers to interrupt a continuous timer when triggered by falling and rising edges of the signal respectively. We then used this information to calculate a duty cycle according to the pseudocode and flowchart below:



```
//having set a timer A2 base to run at 375kHz in continuous mode, with a rising edge capture interrupt for CCR3, falling edge interrupt for CCR2, and an overflow interrupt, calling some ISR function
//global uint32_t variables timerCount, fallingedgepos, initialized to 0
//global float variable duty representing duty cycle, initialized to 0
void ISR{
     if overflow interrupt
          clear interrupt;
          timerCount += $2^{16}$;
//keep track of # of timer counts passed since last rising edge

     else if CCR3 interrupt //rising edge
          duty = $\frac{fallingedgepos}{timerCount + current\ timer\ count}$;
//find duty cycle from time in between falling edge and first rising
//edge divided by time in between 2 rising edges
          timerCount = -current timer count;
//keep track of time passed with respect to last rising edge

     else if CCR2 interrupt //falling edge
          fallingedgepos = timerCount + current timer count;
//get time in between current falling edge and last rising edge

}
```

We will now use this duty cycle to calculate a differential speed to be added to the wheel target speeds (a very slow 6.25% duty cycle). We have implemented the integral control from previous labs to keep the wheels accurately moving at the specified speeds. We will specify speeds based on the calculated differential speeds:

$$\texttt{diffspeed} = \frac{duty * 850 - 450}{400} * 12.5;$$

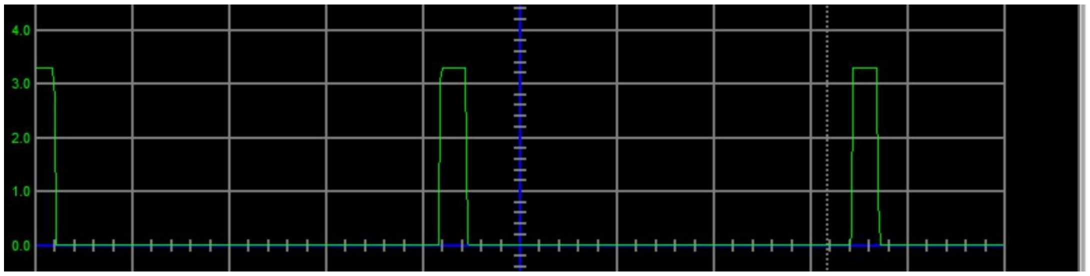
This way we account for the minimum PWM signal being 50/800 duty cycle. Although a duty cycle of 100% yields slightly less magnitude of a differential speed than 0%, this was enough to control the general direction of the car, with the potentiometer needing to be turned slightly right for the car to move straight. We then scale this number ranging from [-1, 0.875] by the maximum magnitude differential speed, in this case 12.5. This lets the car turn with a maximum difference in duty cycle between the two wheels of 25/800. The target speeds of each wheel to be set by the integral control are then modified to represent this difference in speed:

```
target_l = targetSpeed + diffspeed;   //left wheel
target_r = targetSpeed - diffspeed;  //right wheel
```

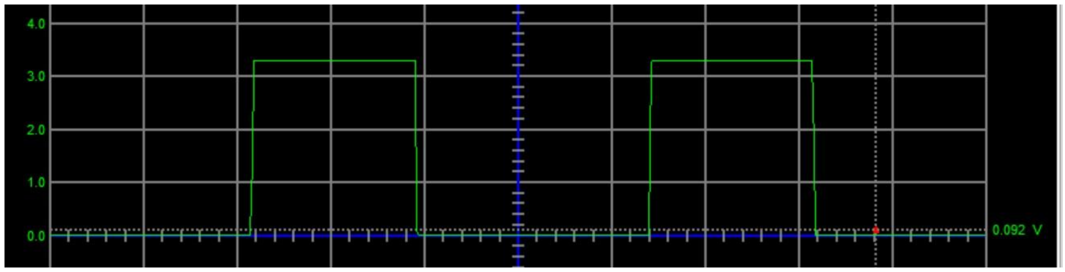
Verification of Operation:

The functionality of the sending and receiving of the infrared signals can be verified by looking at the oscilloscope graphs for the comparator output for certain potentiometer orientations:

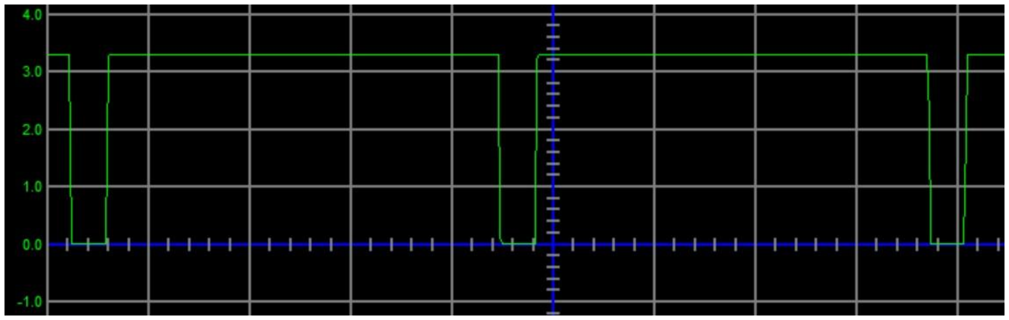
Potentiometer turned left: low nonzero duty cycle expected



Potentiometer turned straight: 50% duty cycle expected



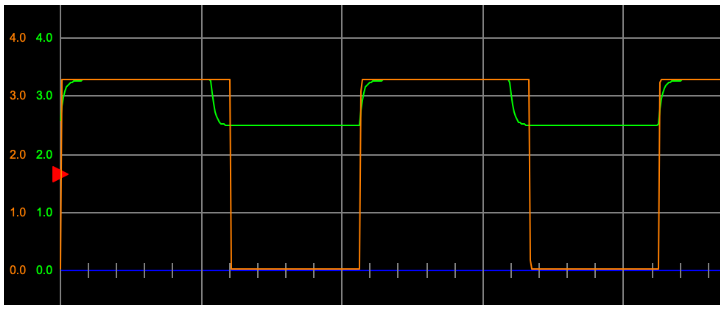
Potentiometer turned right: high, less than 100% duty cycle expected



Seeing the correct corresponding measured duty cycle corresponding to potentiometer input, coming from the comparator output, we can verify that the entire processes for sending and receiving the infrared signal to communicate direction between microcontrollers worked exactly as planned.

We can also verify the amplified comparator signal with respect to the low signal straight from the phototransistor:

Signal from phototransistor (green), Signal from comparator(orange)



Here we see that the phototransistor signal varies from about 2.5-3.3V, all of which is around the range of logic high, meaning the signal would not be recognized from the microcontroller pin. When feeding this signal into the comparator and adjusting the threshold voltage with a potentiometer, we see that we can convert this small difference in voltage to a clear difference between logic 0 (0V) and logic 1 (3.3V) with no values in between.

Conclusion:

Looking back after completion of this project, there were many challenges and bugs that arose. One unexpected behavior we encountered was that the voltage level of the signal coming from the phototransistor did not vary enough to register distinct low and high values on the microcontroller for many cases as described above. Thankfully Dr. Wilt suggested we use a comparator which provided an adequate solution to this problem. We were still limited by the distance at which we could steer the car, but we were able to manage to control the car from a few centimeters away from the phototransistor and still successfully complete the maze. We unsuccessfully tried multiple means to measure the incoming PWM signal from the remote. We encountered mostly logical errors in our process from an incomplete understanding of how the timer capture registers detected input. Our final solution was able to correctly calculate the exact pulse width of the signal, although our use of 2 separate pins and capture registers for rising and falling edges was unnecessary. As Dr. Wilt mentioned, we could have just read the GPIO input pin value to determine whether the edge was rising or falling. We also did not implement a perfect user friendly system to control the direction of the car, which could easily have been fixed by sourcing possible PWM ranges from 1 to 799 out of 800 insead of 50 to 799, although our solution was already able to control the direction more accurately than was needed to solve this maze. In the end, we definitely were not expecting to encounter so many challenges, but overcoming these challenges was rewarding. We were able to further our knowledge of embedded systems as well as circuitry through completing a complicated, fully wireless solution to the proposed problem.

Video demonstration of functionality to give TA's a better understanding of physical limitations to remote control:
https://youtu.be/iWptPetttyM