AME587 Final Project:
One-Dimensional Ultrasound Guidance via Q-Learning Simulated and Trained Tables

By: Mohammad Ghufran and Julio Trejo

Contents

Project Description

As a continuation of a prior project, this project aims to connect generated Q-tables (both randomized and simulated) to a track system that uses actions from the Q-table (left, right, and stay) to guide a person to a designated area on the track. Modulation of 2 motors provides the guidance towards the target, generally with the left motor activation indicating leftward movement, right motor activation indicating rightward movement, and no motors indicating no movement. The Q-tables improve their guidance over time with a reward system, taking in the current track location and comparing it to the prior location.

Applications of this system include instructing various individuals on proper techniques for sports, such as swinging a bat or throwing a football. While this project uses an ultrasound sensor as its primary input, other sensors could take its place for the mentioned applications. One such sensor could be an accelerometer to tell a person how fast a movement should be and, with more dimensions included what directions to move.

The project aims to train a machine to learn how a user behaves for the command it sends. The project is based on the Q- Reinforcement learning technique. Our initial step involves populating a random Q-Table, from which actions can be selected. The table's rows correspond to States, while columns represent Actions.

In this particular case, we have set the epsilon value to 1, which prompts the model to exploit its environment rather than explore it. An action is chosen according to the highest value within a specific row. As a point of reference, refer to the table provided.

State/Action	Action 1	Action 2	Action 3
State 1	Q_{11}	Q_{12}	Q_{13}
State 2	Q_{21}	Q_{22}	Q_{23}
State 3	Q_{31}	Q_{32}	Q_{33}
State 4	Q_{41}	Q_{42}	Q_{43}
State 5	Q_{51}	Q_{52}	Q_{53}
State 6	Q_{61}	Q_{62}	Q_{63}
Continued...	Continued…	Continued…	Continued…

For State S, Action A is determined by the row's maximum Q value. For instance, if we are in State 5 and Q53 has the highest value among Q51, Q52, and Q53, the machine will select Action 3. Based on the chosen action, the machine receives a reward. The Q values corresponding to the state and action are then updated using the following formula:

$$\mathrm{Q}(s_t,a_t) \leftarrow \mathrm{Q}(s_t,a_t) + \alpha * [R_{t+1} + \gamma * \max_a \big(Q(s_{t+1}, a)\big) - Q(s,a)]$$

This process is repeated until either the target for the episode is reached or approximately 60 iterations are completed. After completing roughly 50 episodes, we visualize the initial Q-Table and the final Q-Table with MATLAB's bar3 command. By comparing these plots, we can observe how the Q-Table evolves over the course of 50 episodes.

Physical Setup

The primary components for our system consist of the breadboard with the PIC, motor driver, ultrasound sensor, and RS232 connector. These components are placed atop a movable, user-controlled device seen in Figure 1.

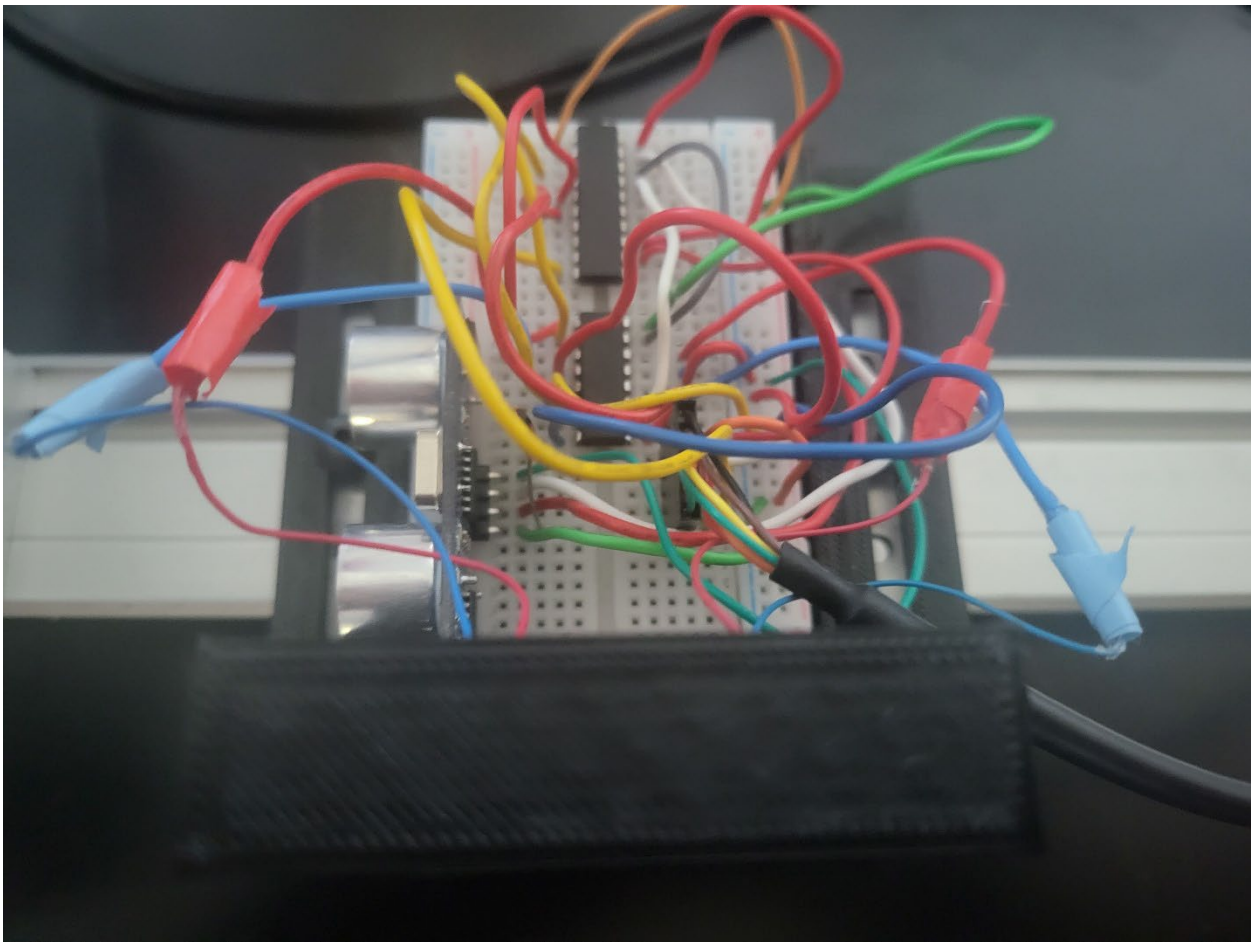


Figure 1: user-controlled device; contains the electrical circuit and allows the user to move the device as needed due to four low-friction wheels interacting with the track beneath it.

To interact with the user, the electrical system utilizes two motors (Figure 2). The possible signals produced by the electrical system include buzzing the left motor, buzzing the right motor, buzzing both motors, and buzzing no motors. Buzzing both motors has been left included in the code but is largely unused as no motors buzzing is a better signal for users.

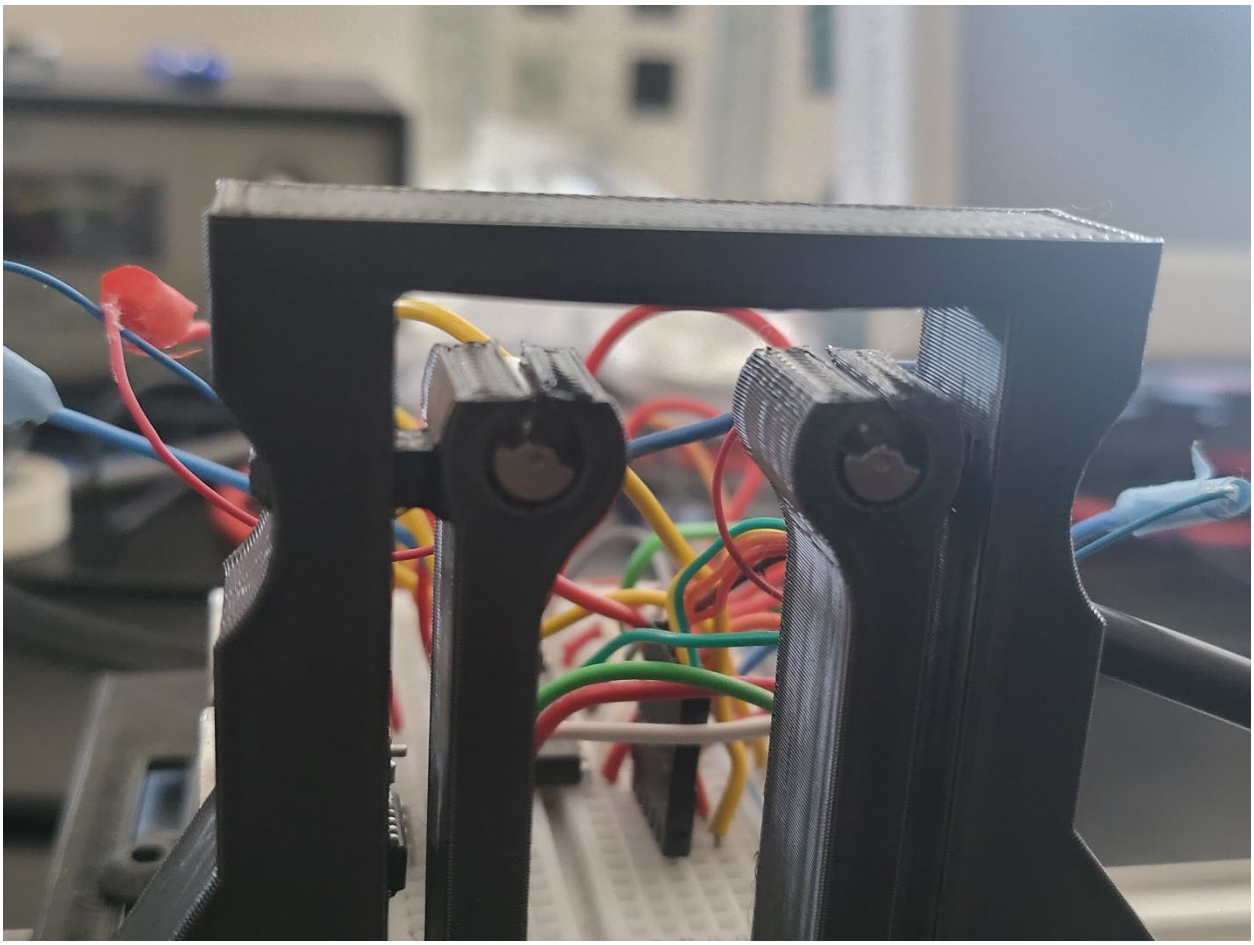


Figure 2: forward view close-up on the motors; each motor is capable of varying strength levels according to the value sent from MATLAB.

The track and board (Figure 3) compose the 1D system explored by the user. The ultrasound sensor communicates the position of the user-controlled device based on an echo pulse emitted by the sensor and reflected of the board back to the sensor. The track allows for definition of states within the MATLAB program such that part of the Q-table is trained based on the location of the user-controlled device in whole inches.

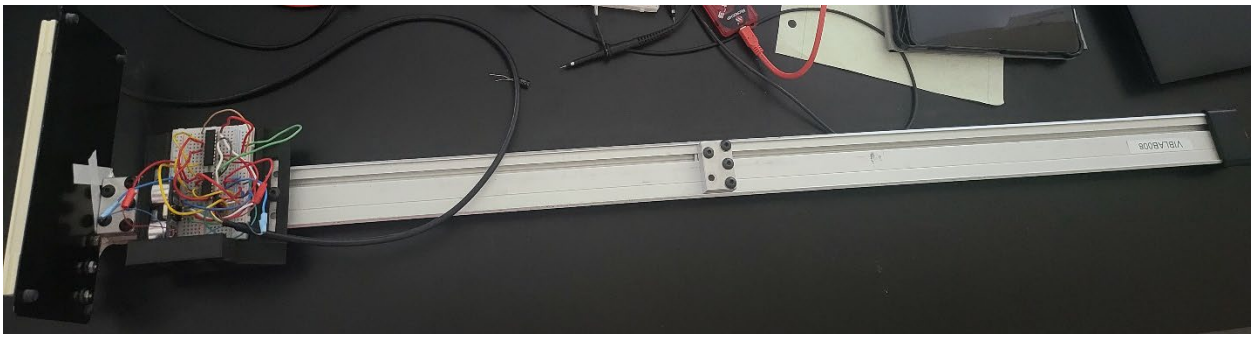


Figure 3: top view of the entire system; the board on the left end acts as a reference point for the ultrasound sensor, and the track itself is currently limited to a range of 5-28 inches, corresponding to 23 states on the Q-table.

Electrical Components

Sensors

1. Ultrasound Sensor (HC-SR04) – This sensor provided the location of the user-controlled device. An echo pulse is emitted when the trigger pin is signaled,
   1.1. Operating Voltage: 5V DC
   1.2. Operating Current: 15mA
   1.3. Measure Angle: 15°
   1.4. Ranging Distance: 2cm - 4m

Figure 4: pin layout for the ultrasound sensor; VCC is for a 5V supply, Trig starts an ultrasound echo pulse when activated, Echo tells when the echo pulse has returned to the sensor, and GND is standard ground.

Actuators

1. Motors () – These actuators communicate the motor driver's pulse-width modulation (PWM). Based on communicated PWM values, one, both, or none of the motors will activate, creating vibrations for the user to sense.
2. Motor driver (L293B) – This actuator applies the PWM signals from the microcontroller to the motors.

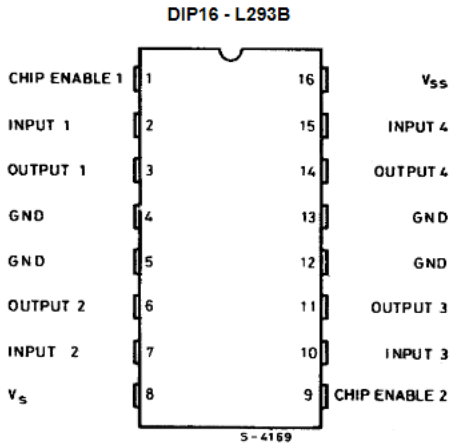
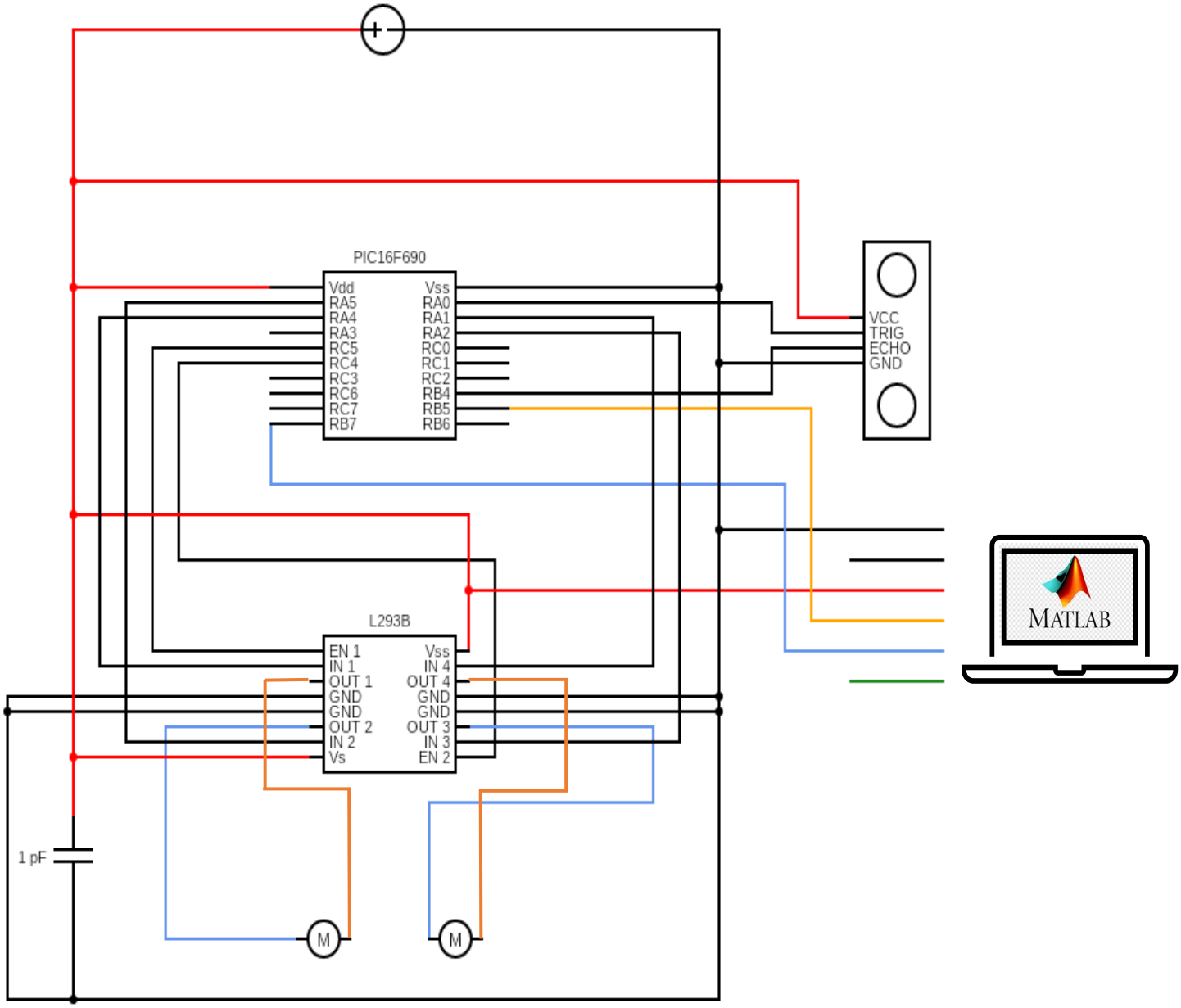


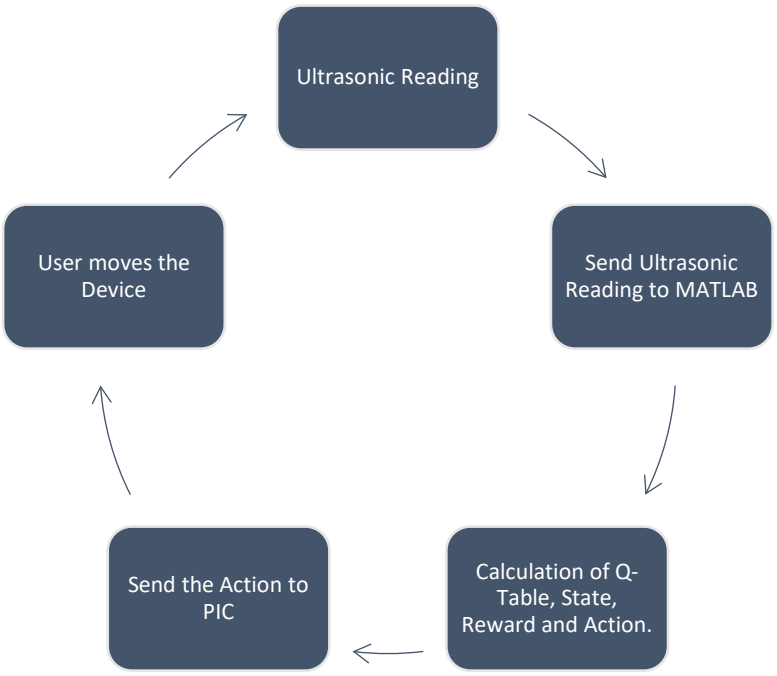
Figure 5: pin layout for the motor driver; enable pins take in the PWM from the microcontroller, VSS indicates the motor driver's logic voltage supply, VS is the motor driver's supply voltage for the motors, outputs 1 and 2 are for motor 1, outputs 3 and 4 are for motor 2, and GND is standard ground. The inputs 1-4 are ignored for this project but correspond to motor direction control.

Design Overview

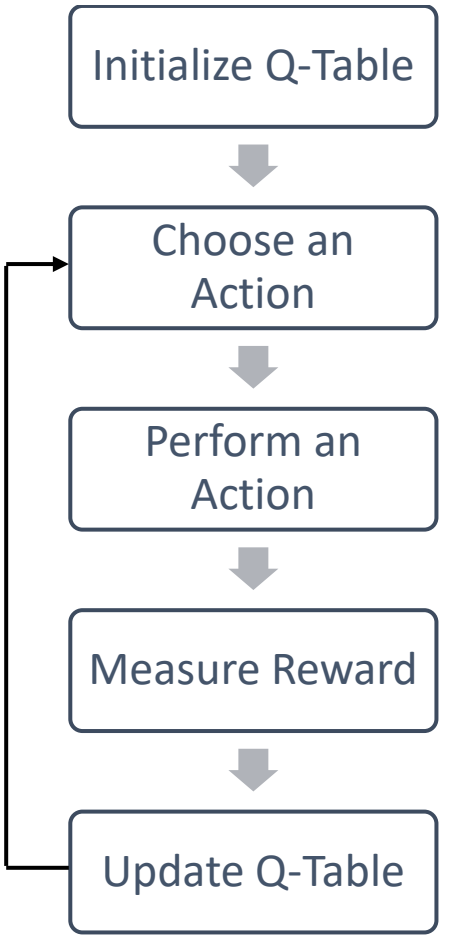
Circuit Schematic



Program Flowchart



Calculation of Q-Table, State, Reward and Action



Special Function Registers Implemented

ANSEL & ANSELH

These registers are used to define the pins on the microcontroller as either digital via a 0 or analog via a 1. All our pins are digital for this project.

OSCCON

This register is used to define the oscillator(s) used for synchronizing operations. The bits used are:

1. SCS – system clock select bit; defines the clock source for the system. 0 sets the source to FOSC<2:0>. 1 sets the source to the internal oscillator. Our project set this bit to 0.
2. LTS – low frequency stable bit; tells if low-frequency internal oscillator has stable operation. 0 indicates not stable while 1 indicates stable. We set this bit to 0.
3. HTS – high frequency status bit; tells if high-frequency internal oscillator has stable operation. 0 indicates not stable while 1 indicates stable. We set this bit to 0.
4. OSTS – oscillator start-up time-out status bit; indicates clock source in use. 0 tells that an internal clock oscillator is in use. 1 indicates that an external clock oscillator is in use. We set this bit to 0.
5. IRCF0-2 – internal oscillator frequency bits; define the frequency of the internal oscillator. We set all bits to 1 for an internal oscillator frequency of 8 MHz.

PSTRCON

This is the power steering register which allows for direct flow of the PWM signal to the necessary pins of P1A-D which corresponds to pins RC5-2. The bits used are:

1. STRA – steering enable bit A; steers the PWM output to P1A. This is used to tell the left motor to activate with the given PWM signal. A value of 1 allows for PWM output. A value of 0 reverts P1A to a port pin.
2. STRB – steering enable bit B; steers the PWM output to P1B. This is used to tell the right motor to activate with the given PWM signal. A value of 1 allows for PWM output. A value of 0 reverts P1B to a port pin.
3. STRSYNC – steering update; tells the system when to update the PWM steering. A value of 0 tells the steering event to occur at the end of a given instruction; this allows for PWM interruption. A value of 1 tells the steering event to happen at the beginning of a PWM period; this results in complete PWM communication each time. We set this bit to 1.

CCP1CON

This is the PWM configuration register which designates the operation of pins P1A-D. The bits used are:

1. CCP1M<3:0> – CCP (Capture Compare PWM) mode select bits; set the specific configuration for P1A-D to react to CCPR1L inputs. The values used are 1100, resulting in P1A-D set as PWM mode and active when high.
2. DC1B<1:0> – PWM duty cycle least significant bits; define the decimal value of a duty cycle. We gave both bits values of 0.

3. P1M<1:0> – output configuration bits; decide how to allocate the PWM to P1A-D. We gave both bits values of 0 for single output mode which provides control over PWM steering to one or multiple of the PWM output pins (P1A-D).

CCPR1L

This is the PWM duty cycle specification register along with DC1B<1:0> of the CCP1CON register. Our system used values from MATLAB to generate the necessary PWM output for the motor driver to activate the motors at varying strength levels from 0 to 255.

PR2

This is the PWM period register that must work together with the timer 2 pre-scale in T2CON. We set this register to 255.

TMR0

This is the timer 0 counter register. The value of timer 0 can be read and written here.

T1CON

This register controls the operation of timer 1. The bits used are:

1. TMR1CS<1:0> – clock source bits; indicates the source for timer 1's clock. We set these bits to 0 to use the internal clock FOSC/4.
2. TMR1ON – timer 1 enable bit; turns timer 1 on or off. 1 is on. 0 is off.

T2CON

This register sets up timer 2 for PWMs. The bits used are:

1. T2CKPS<1:0> – pre-scaler rate bits; defines the timer 2 pre-scaler rate. We set these bits to 10 for a rate of 16.
2. TMR2ON – timer 2 enable bit; enables or disables timer 2. 1 enables. 0 disables. We set this bit to 1.
3. TOUTPS<3:0> – post-scaler rate bits; sets timer 2 post-scaler rate. We set these bits to 1111 for the max rate of 1:16.

OPTIONREG

This register provides the user with the opportunity to adjust certain modules of the microcontroller. The bits used are:

1. PS<2:0> – pre-scaler rate bits; sets the pre-scaler rate for either timer 0 or the watchdog timer (WDT). We set these bits to 010 for a rate of 1:8.
2. PSA – pre-scaler assignment bit; defines whether the pre-scaler rate is for timer 0 or the WDT. 0 is for timer 0. 1 is for the WDT. We set this bit to 0.
3. T0SE – timer 0 counter bit; defines whether timer 0's counter triggers on low-to-high transitions or high-to-low transitions. 0 is low-to-high. 1 is high-to-low. We set this bit to 0.
4. TOCS – timer 0 module clock source bit; defines the source for timer 0's clock. 0 sets the source to the internal instruction cycle clock. 1 sets the source to RA4/T0CKI pin. We set this bit to 0.

TXSTA

This is the transmission status and control register for serial communication. The bits used are:

1. TRMT – transmission shift register status bit; tells if TXREG is empty or not
2. BRGH – baud rate select bit;
3. SYNC – transmission mode select bit;
4. TXEN – transmission enable bit; enables or disables data transmission. 1 enables. 0 disables. We set this bit to 1.

TXREG

This is the transmission data register. Values in this register are sent via serial connection.

RCREG

This is the EUSART receive data register. Data received is placed in this register and must be emptied for further data reception.

RCSTA

This is the receive status and control register which enables received data management. The bits used are:

1. CREN – continuous receive transmit enable bit; enables or disables the data receiver ONLY IN ASYNCRONOUS MODE. 1 enables. 0 disables.
2. SPEN – serial port enable bit; enables or disables serial connection. 1 enables. 0 disables.

BAUDCTL

This is the baud rate control register. The bits used are:

1. BRG16 – baud rate select bit; determines the baud rate of either 8 or 16 bits. 1 is 16 bits. 0 is 8 bits. We set this bit to 0.

SPBRG

This is the free running baud rate timer period register. We set the period to 25.

TRISA, TRISB, and TRISC

These registers determine which pins are outputs with the value 0 and which are inputs with the value 1. We set RA2 and RB4 as inputs and set all other pins as outputs.

INTCON

This is the interrupt control register. The bits used are:

1. T0IF – timer 0 overflow interrupt flag bit; tells whether timer 0 has overflowed or not.

PIR1

This is the peripheral interrupt request 1 register. The bits used are:

1. RCIF – EUSART receive interrupt flag bit; signifies whether the EUSART is full or not. 1 indicates full. 0 means empty.

Theory

Reinforcement Learning (RL) is a field of study focused on decision-making, where an agent learns optimal behavior to maximize rewards in a given environment. This learning process occurs through interactions and observations, resembling how children explore and learn from their surroundings. Without supervision, the agent uses trial-and-error to identify the best sequence of actions, considering both immediate and delayed rewards.

Reinforcement learning can be characterized by the following key components:

- Agent: The decision-making entity that learns to make decisions by interacting with the environment.
- Environment: The external context in which the agent operates, responding to its actions by providing rewards and updating the state.
- State: A representation of the current situation or configuration of the environment, containing all relevant information for the agent to make decisions.
- Action: The choices made by the agent to influence the environment, which can be discrete or continuous.
- Reward: A scalar value provided by the environment as feedback to the agent, indicating how desirable or beneficial an action is in a particular state.
- Policy: A strategy or mapping from states to the agent's actions to decide which action to take in a given state.

There are two main types of RL algorithms: model-based and model-free. Model-free algorithms estimate the optimal policy without using the environment's dynamics, while model-based algorithms use the transition and reward functions for policy estimation.

Q-learning is a model-free, value-based algorithm that updates the value function using the Bellman equation. It contrasts with policy-based algorithms, which estimate the value function through a greedy policy from the previous policy improvement. Q-learning is an off-policy learner, meaning it learns the optimal policy's value independently of the agent's actions. In contrast, an on-policy learner finds a policy that is optimal, considering the exploration steps and learning the value of the policy being executed by the agent. The 'Q' in Q-Learning stands for Quality. Quality here represents a given action's usefulness in gaining some future reward.

Q-Learning Definition

The Q*(s, a) represents the expected value (cumulative discounted reward) of taking action 'a' in state 's' and following the optimal policy. To estimate the value of Q*(s, a), Q-learning uses Temporal Differences (TD) learning. TD learning is a technique that allows an agent to learn from an environment without any prior knowledge of it, by experiencing episodes and adjusting its estimates iteratively. This approach combines aspects of both dynamic programming and Monte Carlo methods.

In Q-learning, the agent maintains a table of Q[S, A], where 'S' represents the set of all possible states and 'A' represents the set of all possible actions. Each entry Q[s, a] in this table corresponds to the agent's current estimate of Q*(s, a), which is updated as the agent interacts with the environment and receives feedback in the form of rewards.

Algorithm

1. Initialize the Q[S, A] table with arbitrary values.
2. Choose an action 'a' for the current state 's' based on the Q[s, a] values, typically using an exploration-exploitation strategy like epsilon-greedy.
3. Perform the chosen action 'a' and observe the resulting state s_{t+1} and reward R.
4. Update the Q[s, a] value using the observed reward and the maximum Q-value for the next state:
   $$\mathrm{Q}(s_t,a_t) \leftarrow \mathrm{Q}(s_t,a_t) + \alpha * [R_{t+1} + \gamma * \max_a(Q(s_{t+1}, a)) - Q(s,a)]$$
   where α is the learning rate and γ is the discount factor.
5. Repeat steps 2 to 4 until a termination condition is reached, such as a maximum number of episodes or convergence of the Q-values.

As the agent accumulates more experience, the Q[S, A] table converges to the optimal Q-values, allowing the agent to select actions that lead to the highest cumulative discounted rewards.

Results

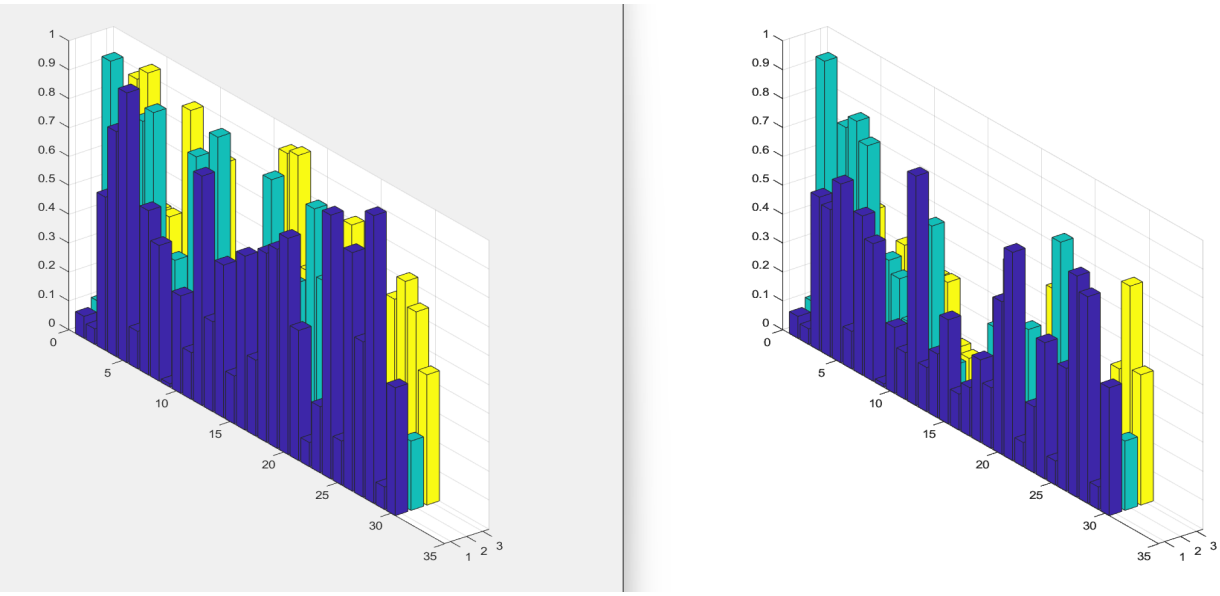


Figure 6: From left: Initial Table and Table after 10 episodes

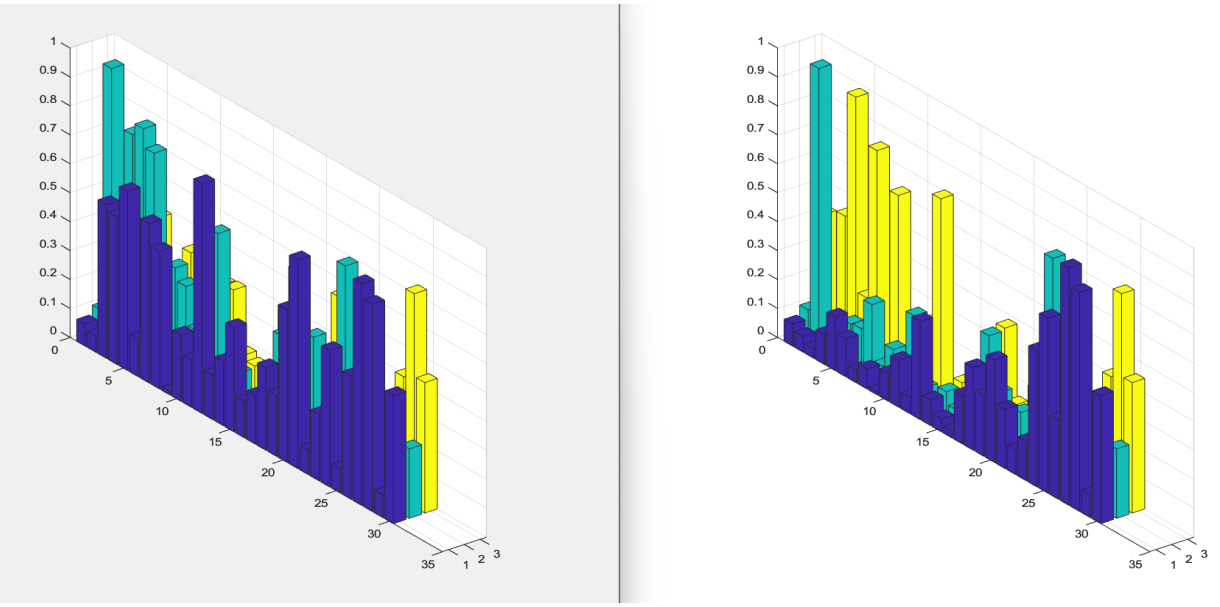


Figure 7: From left: Table after 10 episodes and Table after 20 episodes

Figure 6 and 7 Illustrate the evolution of the Q-table as the model is trained over time, highlighting the changes in Q-values from an initial random Q-Table to 10 episodes and then to 20 episodes.

Initially, the Q-table is filled with random values, reflecting the agent's lack of experience and knowledge about the environment. As the agent begins to explore and interact with the environment, it updates the Q-values based on the rewards it receives and the state transitions it experiences.

After 10 episodes, the Q-table has started to capture the agent's understanding of the environment, reflecting the actions that have been taken and their associated rewards. At this stage , the Q-values may still be far from optimal, as the agent has yet to thoroughly explore the state-action space.

By the time the agent has completed 20 episodes, the Q-table has undergone further refinement. The agent has had more opportunities to explore various actions and states. The agent's performance should improve as the Q-Table becomes more accurate, moving closer to the optimal policy.