

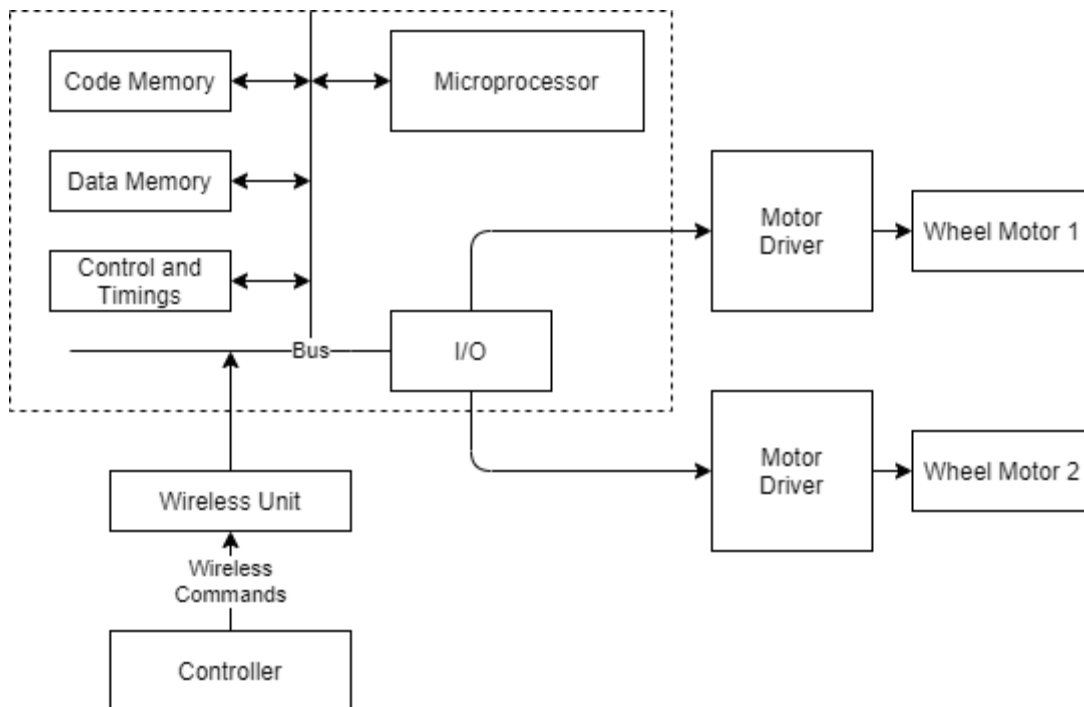
ELM 334 - Homework #1

Ömer Emre Polat
1801022037



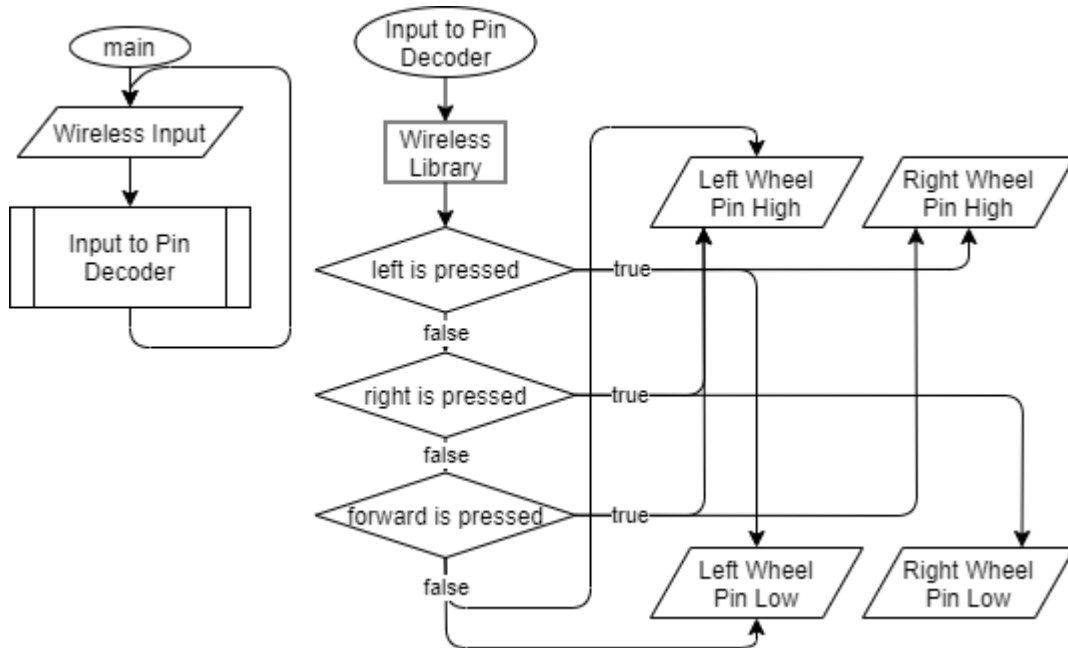
A. Problem 1

Hardware block diagram is given below.



This block diagram is designed to fit the needs of the current task. It does not include a switch to control the battery input and output. Battery supplies power to the microcontroller and motor driver. Wireless unit will likely have a lower V_{in} voltage so it will be supplied from the microcontroller. Pin outputs are controlled by the microcontroller according to the given wireless command. Motor drivers are added because microcontroller probably won't have enough power output from its pins to drive a wheel motor.

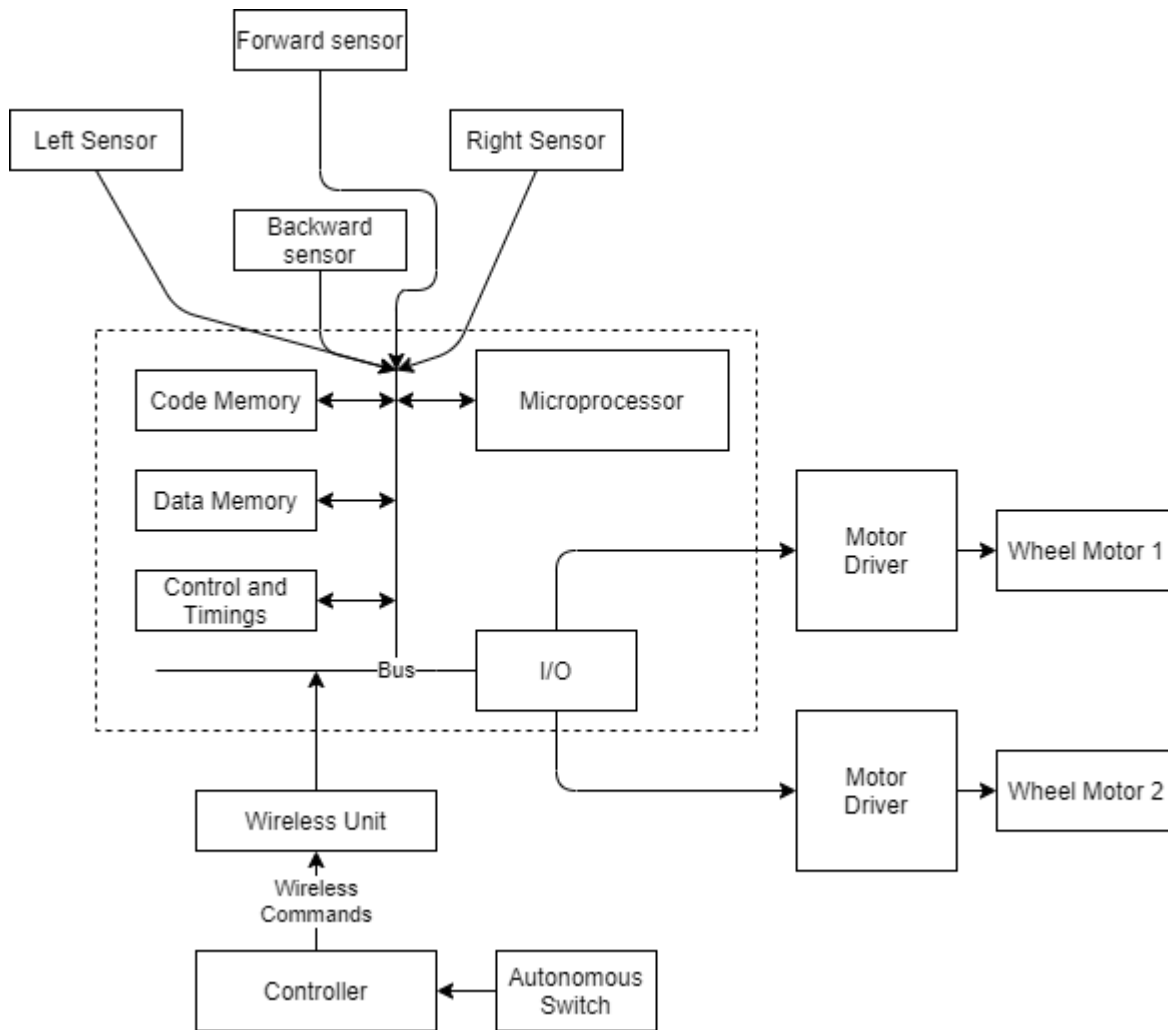
Software flowchart is given below.



Software must ensure that the input from the wireless unit is expected constantly. Attaching two pins to drive the wheels ensures that the wheels will be turned when the microcontroller decodes the commands sent by the wireless unit.

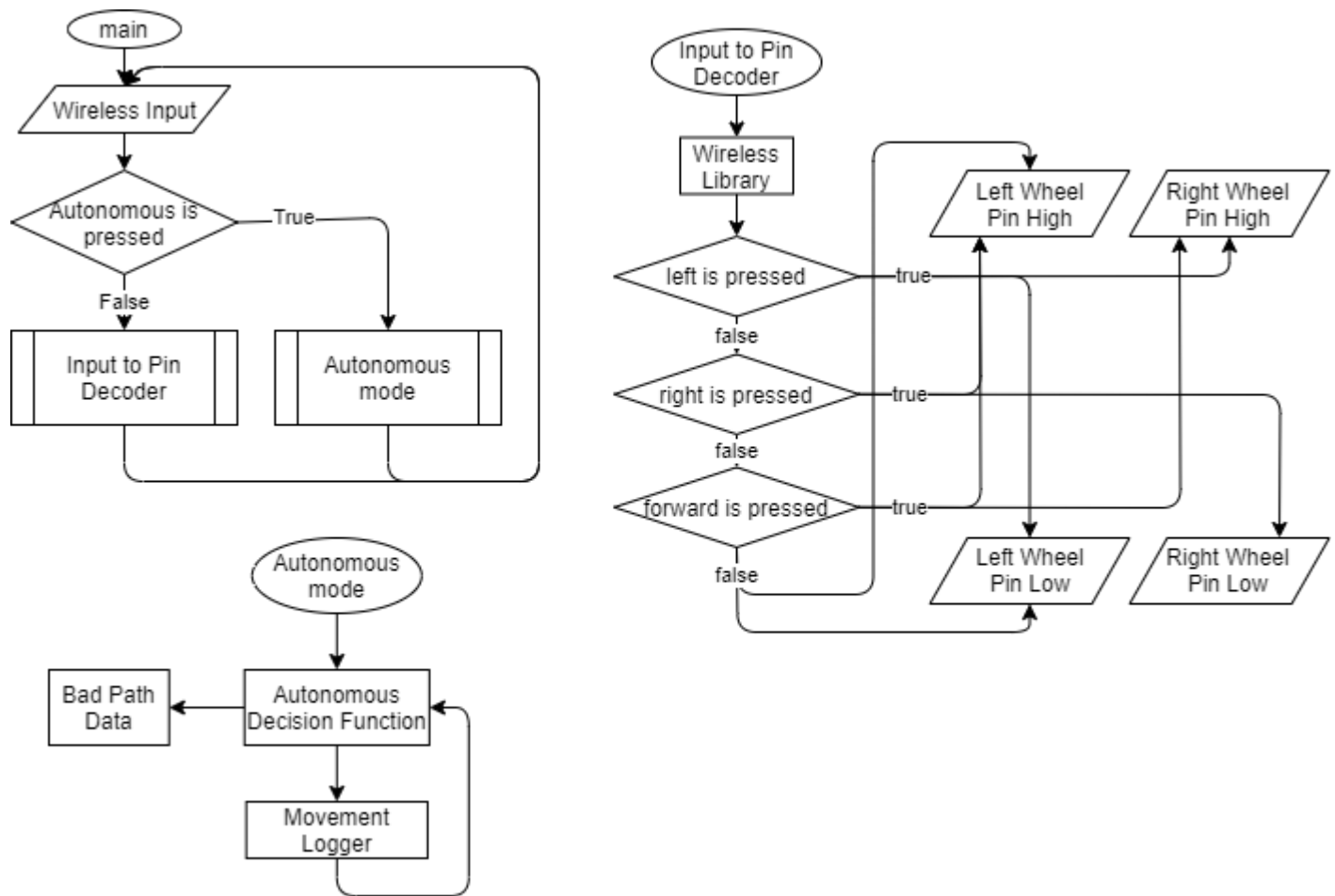
B. Problem 2

Hardware block diagram is given below.



In this hardware block diagram there are added sensors to provide information to the microcontroller for the autonomous mode. Sensors each have their own input pins but they can all be connected together using a bus. Autonomous signal comes from the wireless controller to the wireless unit then to the microcontroller.

Software flowchart is given below.



Software is designed to fit the needs of both autonomous and remote controlled action. Autonomous mode works using a decision making algorithm that takes the current movement logs and bad path data.

C. Problem 3

This Software is designed for logging certain movements of the autonomous drive mode. Code was made in visual studio code.

```

#include <stdio.h>
#include <stdlib.h>

//queue that stores the node array
struct Queue
{
    int head, tail, size;
    unsigned capacity;
    struct Node* array;
};

//node to store coordinate
struct Node
{
    char x;
    int y;
};

//initializes the queue with given capacity of max nodes
struct Queue* initializeQueue(unsigned capacity)
{
    //queue is dynamically allocated using malloc
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    //variables are set into the newly created queue
    queue->capacity = capacity;
    queue->head = queue->size = 0;
    queue->tail = capacity - 1;
    //array is allocated with malloc to store (capacity) number of elements
    queue->array = (struct Node*) malloc(sizeof(struct Node) * queue->capacity);
    return queue;
}

//push function to add a new node to the queue
void push(struct Queue* queue, struct Node item)
{
    queue->tail = (queue->tail + 1) % queue->capacity;
    //setting the node array elements to given items elements
    (queue->array + (queue->tail))->x = item.x;
    (queue->array + (queue->tail))->y = item.y;
    //size is increased
    queue->size = queue->size + 1;
}

//pulls the last element out of the array
//this function can be useful when the autonomous mode needs the backtrack when faced with a dead end
struct Node* pull(struct Queue* queue)
{
    struct Node* item = (struct Node*) malloc(sizeof(struct Node));
    //takes the already existing node element and copies its elements to an item object
    item->y = (queue->array + (queue->head))->x;
    item->x = (queue->array + (queue->head))->y;

    queue->head = (queue->head + 1) % queue->capacity;
    queue->size = queue->size - 1;
}

```

```

    return item;
}

//prints the moves recursively
void print_moves(struct Queue* queue)
{
    printf("Path followed:\n");
    for(int i=0; i<queue->size; i++)
    {
        if(i == (queue->size)-1)
        {
            printf("%c%d", (queue->array + (i))->x, (queue->array + (i))->y);
        }
        else
        {
            printf("%c%d, ", (queue->array + (i))->x, (queue->array + (i))->y);
        }
    }
    printf("\nTotal moves: %d", queue->size);
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main()
{
    struct Queue* queue = initializeQueue(10);

    //create 1 coordinate
    struct Node coord1;
    coord1.x = 'D';
    coord1.y = 8;

    //create another coordinate
    struct Node coord2;
    coord2.x = 'C';
    coord2.y = 8;

    //create last coordinate
    struct Node coord3;
    coord3.x = 'C';
    coord3.y = 7;

    //add the coordinates
    push(queue, coord1);
    push(queue, coord2);
    push(queue, coord3);
    //print the added coordinates
    print_moves(queue);

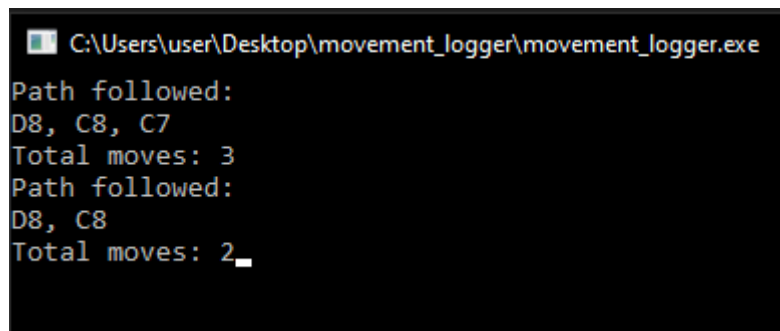
    printf("\n");
    //backtrack one node
    pull(queue);
    //print the backtracked coordinates
    print_moves(queue);
}

```

```
return 0;
```

```
}
```

Code is explained using comments in the code. Working summary is that two structs are created. Queue struct includes the main array where the node info is stored, head of the array, tail of the array, capacity of the array and the size of the array that's actually being used. Node struct only includes two variables one char for x row and an int for the y row. All the other functions are constructed upon these structs. Pull function is added extra for the backtracking of autonomous mode.



```
C:\Users\user\Desktop\movement_logger\movement_logger.exe
Path followed:
D8, C8, C7
Total moves: 3
Path followed:
D8, C8
Total moves: 2_
```

Test code output shows it working as intended. After adding 3 moves and printing we can see 3 moves that are printed on the screen. Then the pull command removes the last element out of the array and only 2 elements are left.