GEBZE TECHNICAL UNIVERSITY

ENGINEERING FACULTY

ELECTRONICS ENGINEERING

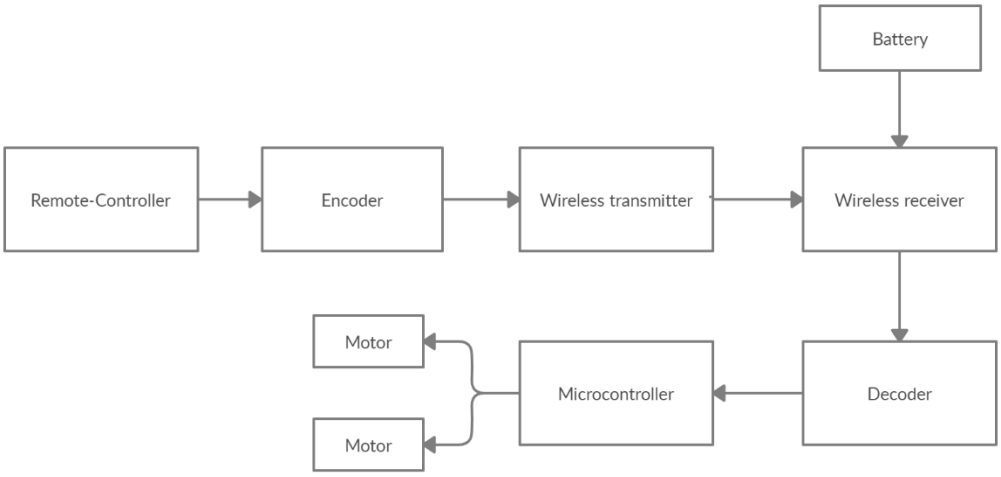
ELEC 334

MICROPROCESSORS

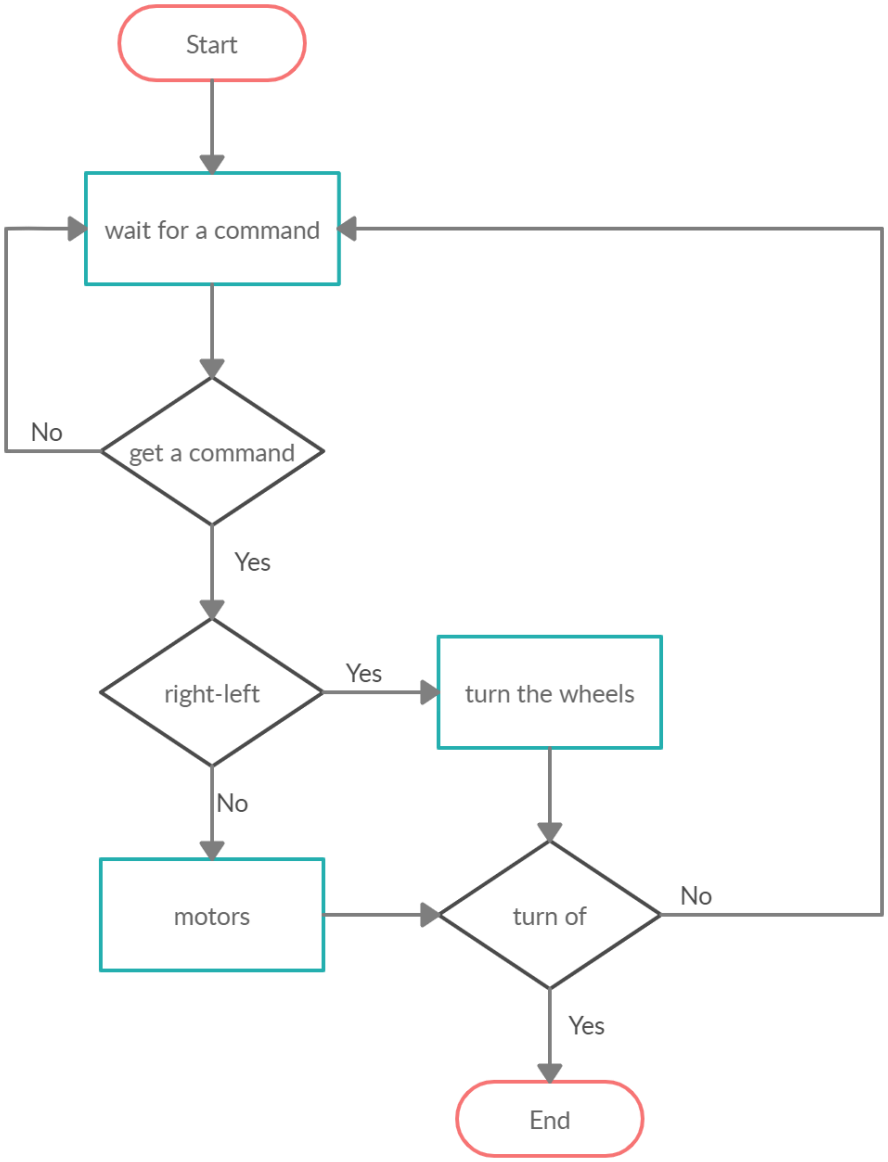
HW1

Name - Surname	Yağmur DERYA
Student ID	171024011

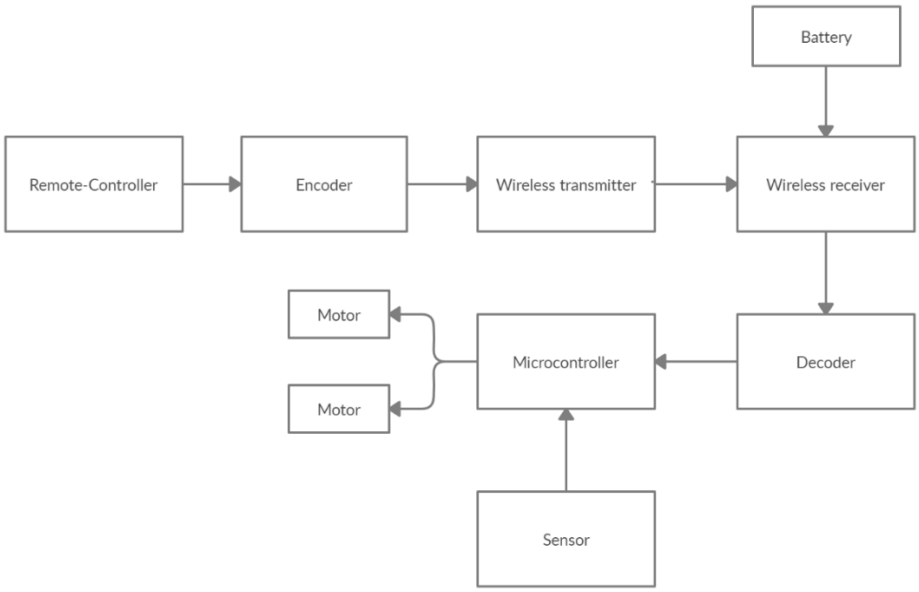
Problem 1. The block diagram of a remote-controlled 2-motor 3-wheel robot:



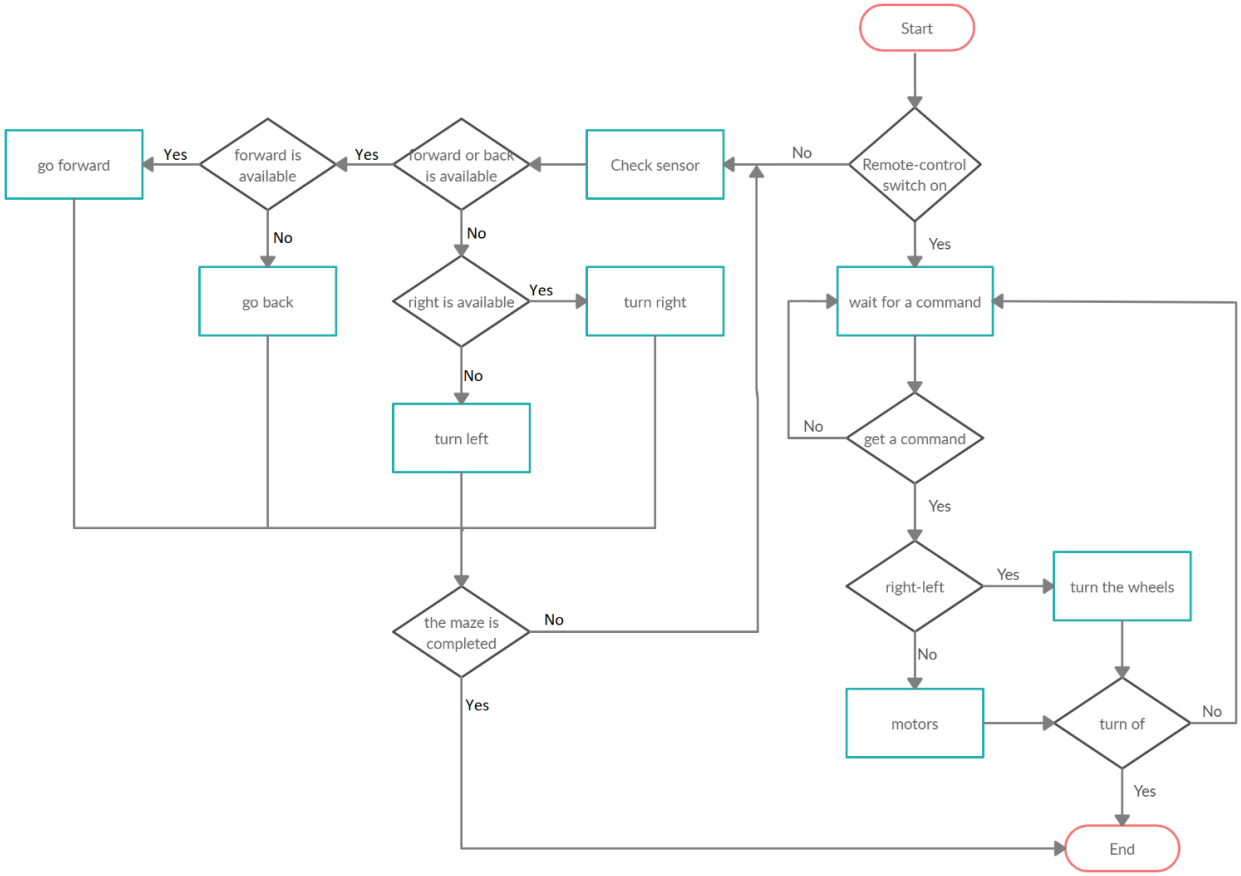
The flowchart of a remote-controlled 2-motor 3-wheel robot:



Problem 2. The block diagram of an autonomous robot which can be used with remote-control functional by using a switch.



The flowchart of an autonomous robot which can be used with remote-control functional by using a switch.:



The sensor might be a radar sensor, which sends out radio waves that detect objects and speed in relation to the vehicle in real time, or a lidar sensor, which uses lasers instead of radio waves, in this problem.

Problem 3. The C code of the logger function that will remember all the locations the robot travelled and counts the number of steps until the robot finishes its move.

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

#define SIZE 100

typedef struct Position{ // The position of the robot in the maze
        int num;
        char let;
} Position;

typedef struct Queue{ // The data structure that keeps the all the previous moves.
        struct Queue *next; // next position
        struct Position* pos; // current position
        int size;
} Queue;

void logger(struct Position moves[], int size);
struct Position* random_moves(struct Position moves[], int size);
struct Queue* createQueue(struct Position* p);
void push(struct Position* next_p, struct Queue* current);
void print_moves(struct Queue* head, int size);

int main(){
        srand((unsigned)time(NULL)); // srand() makes use of the computer's internal clock to control the
choice of the seed, by this way random values always change

        int size = rand() % 50 + 1; // asssigns random size values for moves array with a top limit which is 50
so we can check easily.
        struct Position moves[size]; // random_moves
        random_moves(moves, size);

        /* // to check if the print_moves function works true
        int i;
                for(i=0; i<size; i++){
                printf("%c%d-", moves[i].let, moves[i].num);
        }
        printf("\n");
        */

        logger(moves, size);

        return 0;
}

void logger(struct Position moves[], int size){

        struct Position* first_p = (struct Position*)malloc(sizeof(struct Position));
        first_p->let = moves[0].let;
        first_p->num = moves[0].num;
        struct Queue* queue     = createQueue(first_p); // assigns first position to queue

        struct Queue* head = queue; // keeps head of the queue in order to not lose head

        int i;
        for(i=1; i<size; i++){
                struct Position* p = (struct Position*)malloc(sizeof(struct Position));
                p->let = moves[i].let;
                p->num = moves[i].num;

                push(p, queue);
```

```
                queue = queue->next;
                queue -> next = NULL; // clears next for the next loops for any problems
        }

        print_moves(head, size);

}

void print_moves(struct Queue* head, int size){
        printf("Path followed:\n");
        int i;
        for(i=0; i<size; i++){
                printf("%c%d", head->pos->let, head->pos->num);
                if(i != size-1){
                        printf(", ");
                }
                head = head->next;
        }
        printf("\nTotal moves: %d", size);
}

struct Queue* createQueue(struct Position* p){
        struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
        queue->size = SIZE;
        queue->next = NULL;
        queue->pos = p;
        return queue;
}

void push(struct Position* next_p, struct Queue* current){
        current->next = (struct Queue*)malloc(sizeof(struct Queue));
        current->next->pos = next_p;
}

struct Position* random_moves(struct Position moves[], int size){
        int i;
        char letters[8] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'};

        for(i=0; i<size; i++){
                moves[i].num = rand() % 8 + 1; // numbers star from 1 to 8 so we need to add 1 to not get 0
                int l = rand() % 8; // gets random value between 0 to 7.
                moves[i].let = letters[l]; // gets a random letter from letters array, l used as an index.
(letters[0] = A, letters[5] = F etc.)
        }

        return moves;
}
```