Coin Picking Robot 2025

University of British Columbia

Electrical and Computer Engineering

ELEC_V 291/292 201 Winter 2024-2025

Instructor: Dr. Jesus Calvino-Fraga P.Eng

Lab Section L2B: Tuesday and Thursday 8:00 am to 11:00 am

Project 2- Coin Picking Robot

Group #: B07

Group Members	Student number	Percentage contribution	Signature
Bangyan Jiang	72488521	102	
Xuanyu Bao	19074525	102	
Pengrui Zhao	38972659	100	Pengrui Zhao
Emma Zhu	51350791	102	
Harris Liu	45513629	102	
Yi Hou	60556305	92	

Date of Submission: April 8, 2025

Table of Contents

1. Introduction

1.1 Objective

Key objectives of this project include designing, building, programming, and testing a coin-picking robot. The robot needs to have two modes: Automatic, where it detects and collects 20 coins within a wired AC-defined perimeter, and Manual, where it's controlled remotely by an operator. The microcontroller used in the robot and the operator need to be in a different microcontroller system and the robot and the operator must be battery operated.

1.2 Specifications

1.2.1 Hardware information

1. **Circuit specification for Robot**

- EFM8 microcontroller: 8051 Family.
- Radio circuit: JDY-40 with three pins connected, TXD, RXD, and SET. The JDY-40 module is powered by 3.3V, which is regulated from a 5V supply using an MCP1700 voltage regulator.
- Wheel circuit: H-bridge circuit using LTV846 and four N-MOSFET and P-MOSFET.
- Servo circuit: LTV846 with 3.3k ohms resistors.
- Magnetic circuit: N-MOSFET with diode.
- Metal detector: One inductor connected to the bottom of the car, tow capacitor, one N-MOSFET and one P-MOSFET.
- Perimeter detector: Inductor and capacitor connected to the bottom of the car, along with an LM358-based peak detector circuit built using 10 µF capacitors and resistors.

- Distance detector: HC-SR04 distance detector, DFPlayer mini with speaker.
- Bluetooth module HC-05: Connect to the same TXD and RXD as JDY40, use a switch to control.
- LM7805: Step down 9 volts battery supply to 5V.

2. Robot Construction

- Solarbotics GM4: Gear Motor 4 - Clear Servo, two 3D printed wheels.
- Tamiya 70144: Ball Caster.
- Battery: 4 x AA with battery holder, and one 9V battery with battery clip.

3. Circuit specification for Remote controller

- STM32 microcontroller: ARM family.
- LCD and Buzzer: LCD powered by 5 volts. A buzzer connected to a N-MOSFET.
- BO230XS USB adapter.
- Radio circuit: JDY-40 with three pins connected, TXD, RXD, and SET. The JDY-40 module is powered by 3.3V, which is regulated from a 5V supply using an MCP1700 voltage regulator.
- Button: Three push buttons for choosing mode and starting the program.
- Joystick: PS2 joystick connected to the ADC input pins of STM32.

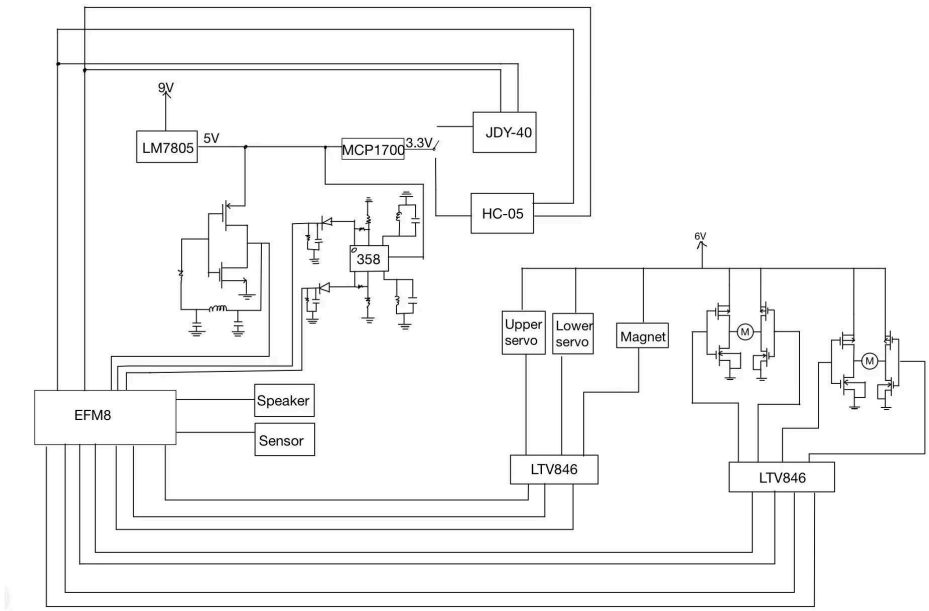


Figure 1: Car hardware block diagram

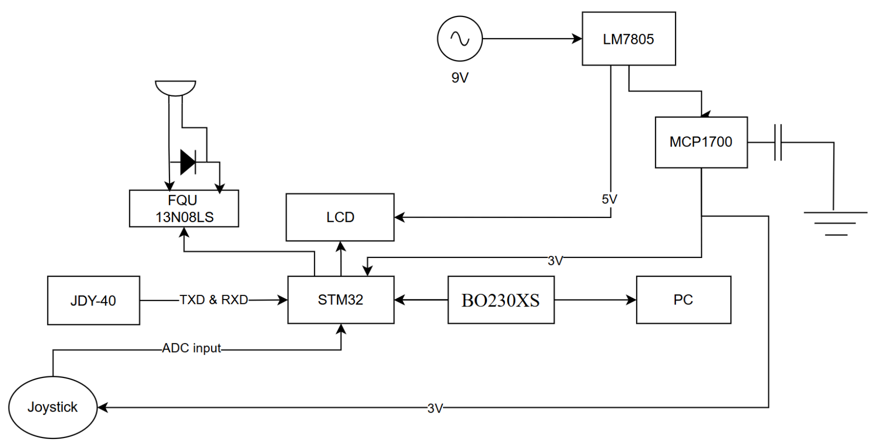


Figure 2: Remote hardware block diagram

1.2.2 Software information

- Programming language: Programmed in C language, the recording and plotting of the car's trajectory were implemented using Python programming.
- Using makefile to link several files: adc, lcd, UART2.

- Motor and Servo control, Period measurement in Timer 5 interrupt.

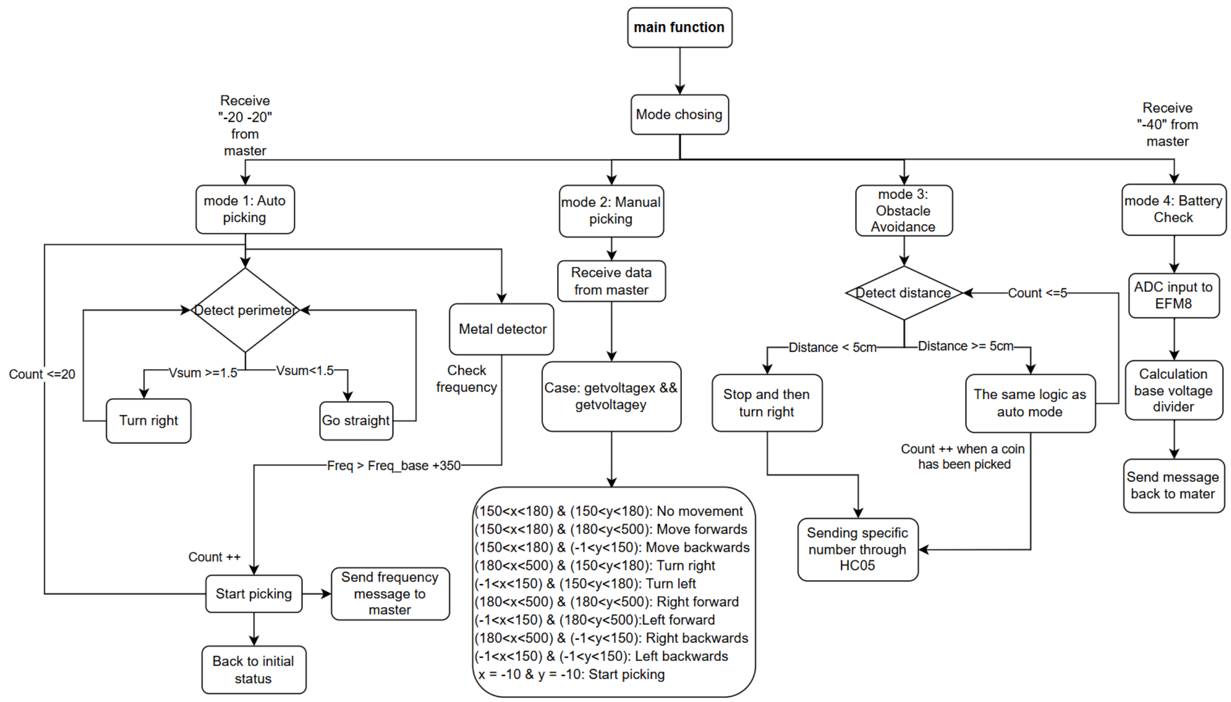


Figure 3 : Car software block diagram

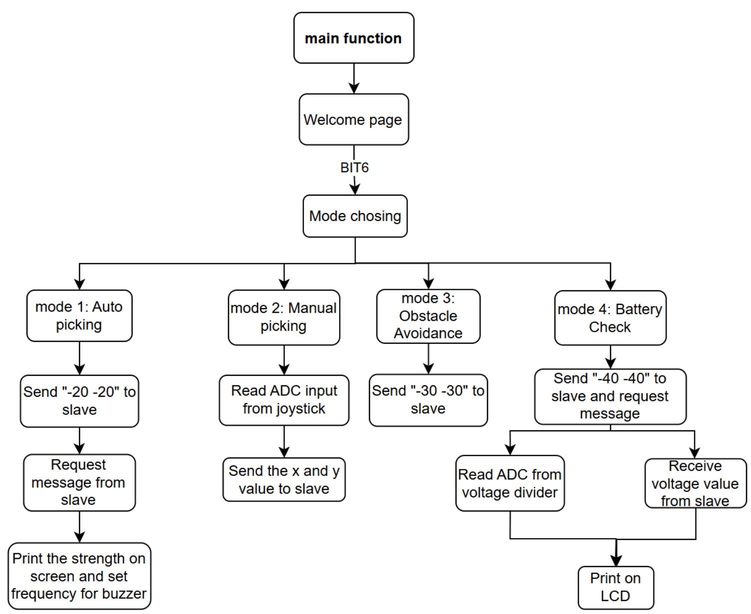


Figure 4: Remote software block diagram

2.Investigations

2.1 Idea Generation

Our team began the design process by first analyzing the functionalities required for both the robotic car and the remote controller. We also conducted research on various microcontrollers that could be used for this project. We proposed using a Colpitts oscillator with an LM358-based peak detector for metal and perimeter detection, and a coin picking mechanism using servo motors and an electromagnet. Moreover, we also considered several bonus features. Automatic obstacle avoidance was proposed after recognizing that the robot may encounter unexpected obstacles while operating autonomously, and battery voltage monitoring was added to allow users to check battery levels directly from the controller or robot without needing external tools. Our team also wanted to add a function that can trace our car, so we considered adding another bluetooth module to send data to the PC.

2.2 Investigation Design

To validate our design decisions, our group conducted a series of investigations involving component testing, signal analysis, and experimental measurements. We began by gathering information from datasheets, documentation, and past projects. After comparing six microcontrollers based on performance, I/O availability, programming ease, and peripheral compatibility, we selected the STM32 for the remote controller and the EFM8 for the robotic car.

To validate the coin detector, we tested various capacitor combinations, and analyzed frequency changes for coin differentiation. For the JDY module, we performed signal reception tests across multiple channels to assess strength, stability, and interference, ultimately selecting the most reliable one for communication. For the perimeter detection module, we evaluated the ADC inputs on the EFM8 chip, as prior usage raised concerns about analog accuracy. Testing helped us identify stable pins for perimeter detection.

These targeted investigations provided essential data that informed our design decisions and improved system reliability.

2.3 Data Collection

To verify that the frequency data from the metal detector circuit were accurate, our team used an oscilloscope to observe the waveform output from the metal detector circuit. By placing coins near the inductor, we expected a small shift in the frequency or shape of the waveform, which could be visually confirmed on the oscilloscope. This method allowed us to determine which capacitor should be used to generate the best performance.

For the perimeter detection system, we supplied known voltages to the ADC input pins of the EFM8 microcontroller using a voltage source. We then used our embedded code to read the ADC values and compared the digital readings against the expected values. This helped us evaluate the accuracy and consistency of each ADC pin, especially considering that our EFM8 chip had been used extensively and may have experienced degradation over time.

To accurately determine the robot's trajectory in our footprint tracking system, our team collects a group of structured data with various precise measurement techniques. Initially, we measured the total length of the experimental field using tape measures. Subsequently, we recorded the robot's traversal time across the entire field in multiple trials, carefully timing each run with digital stopwatches to ensure accuracy and repeatability. Recognizing the influence of battery voltage on motor performance, we further collected velocity data at multiple voltage levels, specifically at increments between 6V and 5V. We monitored the voltage using voltmeters, while simultaneously recording the vehicle's speed with the same approach we mentioned before. By averaging velocity values measured under these varying conditions, we obtained a representative speed value reflective of realistic operational scenarios[8.1]. Since the robot's speed during turning differs from that during straight-line motion, we measured the turning angle and the time taken for each turn to calculate the angular velocity. These carefully collected dataset provided a solid basis for subsequent data synthesis and analysis, enabling accurate trajectory mapping and further design optimization.

2.4 Data Synthesis

To synthesize the collected data and draw meaningful conclusions, our team had systematic analysis techniques across different subsystems of the robot.

For the metal detection system, we compared the frequency readings obtained from the inductor circuit with and without the presence of metal objects. By analyzing the frequency shifts observed on the oscilloscope, we determined the optimal capacitor value for reliable coin detection. Additionally, we applied a moving average filter to smooth out noise, ensuring consistent and accurate detection thresholds.

In the perimeter detection system, we synthesized ADC readings from the EFM8 microcontroller by comparing measured digital values against known input voltages. We repeated this process for each pin in EFM8 to assess the accuracy and potential degradation of the ADC pins due to prolonged use. Variations in readings were accounted for by calibrating the system to maintain a more reliable boundary detection.

For our footprint tracking system, we synthesized data from distance, time, and angular change to calculate the robot's motion trajectory. First, we measured the length of the field and recorded the time the car took to travel across it. Using the formula *Speed = Distance / Time*, we obtained the car's average linear velocity.

Next, we collected data on the turning angle and duration, which allowed us to compute the robot's angular velocity. By combining the linear velocity and turning angle over time, we were able to reconstruct the robot's trajectory.

We also investigated how battery voltage affects speed. By measuring the car's speed under five voltage levels ranging from 6V to 5V, we observed slight variations and calculated an average speed for trajectory estimation[8.1]. Noting that the relationship between battery voltage and motor speed is nonlinear, we identified this as a limitation and a potential area for improvement. In future iterations, we propose real-time monitoring of battery power via Bluetooth to dynamically adjust speed calculations for higher accuracy.

2.5 Analysis of Result

To validate the system's performance and conclusions, we systematically evaluated observed errors, theoretical limitations, and measurement uncertainties as follows:

a. **Metal Detection System**
   i. **Error Assessment:** The inductor-based detector exhibited frequency fluctuations of ±400 Hz due to electromagnetic interference, causing occasional false positives.
   ii. **Calibration:** Implemented a moving average filter with 10-sample window size, reducing detection errors to ±200 Hz under stable conditions.

b. **Perimeter Detection System**

   i. **Error Assessment:** ADC readings showed ±0.5 mV deviations compared to reference voltages, attributed to aging microcontroller pins.

   ii. **Calibration:** Applied offset compensation, maintaining accuracy within ±0.2 mV for boundary detection

c. **Error Mitigation and Validation**

   i. Successful coin retrieval in 19/20 trials confirmed metal detection reliability

   ii. Trajectory errors remained below 10% of total path length

Through iterative testing and calibration, we confirmed the system operates within design specifications, with residual errors acceptable for its intended application scope. Future improvements would focus on encoder-based positioning and real-time voltage compensation to further reduce uncertainties.

3.Design

3.1 Use of Process

We followed the general engineering design process to develop our system. By carefully going through the project constraints and objectives, our team tackled the complex problem of coin picking by breaking down the design into distinct modules such as motor control, sensor integration, servo actuation, and wireless communication. We first defined clear requirements and then iteratively developed and tested individual components. After testing all the individual components were working properly, our team started to assemble all the elements together. Then we tested again the whole circuit part by part ensuring we assembled correctly. By modularizing tasks, documenting each subsystem, we were able to integrate these components into a cohesive, robust solution that meets the project's objectives.

3.2 Need and Constraint Identification

Our team began by gathering all project specifications and then holding group discussions to identify the core needs. We identified electrical engineering students and companies in electrical design as our primary stakeholders. To meet the stakeholders' requirements, we identified several key functions for the robotic car and remote controller: The robot needed the capability to pick coins both manually, and automatically, real-time responsiveness needs to be provided. In an automatic mode, the system was required to accurately identify and pick up all denominations of Canadian coins, minimizing detection errors. At the same time, when facing a variety of obstacles, the robot had to quickly detect, analyze, and respond to various obstacles while effectively tracking its trajectory. Our project includes several constraints. Both systems must be battery-powered, requiring careful power management to ensure stable performance over time. Wireless modules such as JDY-40 and WIFI share the ISM frequency band, introducing risks of communication interference. By optimizing our design, we balanced functionality, efficiency, and usability.

3.3 Problem Specification

To better meet the stakeholders' requirements, our group further refined the design requirements by translating these needs into measurable, actionable specifications. Firstly, in manual mode, the remote controller should be able to command the car to move in eight different directions, with speed adjustments made according to ADC readings from the joystick. Secondly, for convenience in auto-picking mode, the number of coins picked should be displayed on the LCD board, with different frequencies used to indicate various coin strengths. Thirdly, in auto-picking mode, the robotic car should automatically stop after picking up 20 coins without requiring any control signals from the remote controller, while also being capable of transitioning to other states after stopping. Additionally, to ensure that coins are picked up precisely, the range of motion for the servo should be expanded so that coins located at the corners or edges of the robotic car can also be picked up smoothly.

3.4 Solution Generation

We used the ADC readings with different duty instead of just simply setting the signal for wheels motions to high or low to control wheels moving. Also in the manual mode, by applying different forces to push the joystick, the voltage reading varies from approximately 1.7V to 3.5V and 0V to 1.5V, revealing that our speed should change based on the portion of the reading voltages. To improve the reach of the servo arm, we extended its PWM range, enabling the robot to pick up coins at the edges and corners more reliably. For coin detection feedback, the robotic car sends frequency signals via the JDY-40 module to the remote controller. Upon receiving the signal, the controller increments the coin count and updates the LCD. Beep frequency also varies with signal strength to reflect different coin types. These design choices were made after evaluating performance, responsiveness, and overall system reliability.

3.5 Solution Evaluation

We divided the entire project into four distinct modes: manual mode, auto-picking mode, obstacle avoidance mode, and battery reveal mode. In our code, we combined all servo and motor control functions within the Timer5_ISR interrupt, including calculations for period and frequency. However, this integration sometimes leads to unintended behavior of the servo. We attempted to synchronize the reload times for both the servo and the sensor, but this approach did not resolve the issue.

Additionally, while integrating the HC-05 Bluetooth module onto the remote controller, we encountered a problem. If we connected the HC-05's TX and RX pins directly to the same pins used for the JDY-40, the data transmitted by the JDY-40 would be intercepted by the HC-05, thereby interfering with the control of the robotic car. After reviewing the basic functionalities of both the JDY-40 and HC-05 modules, we decided to connect the HC-05 to the robotic car instead. This configuration allows us to send data to our computer for path plotting using Python, without compromising the control provided by the JDY-40 module.

3.6 Detailed Design

Electronic Circuit

The main electrical hardware components of our circuit are a STM32L051 Microcontroller on master and a EFM8 Microcontroller on slave, LM7805, MCP 1700, JDY-40, HC-05, HC-SR04, DFPlayer mini, GM4. Structures we control servos and motors are made up of two main structures, which are H-bridges and Optocouplers. To be spe（9cific, H-bridges include two P-type MOSFETs and two N-type MOSFETs, which directly connect the two terminals of a motor or servo to the positive and negative poles of a 9V battery. It is symmetrical in form, with a N-type MOSFET, a P-type MOSFET and an Optocoupler on the both sides of the motor/servo. The Optocouplers act as an interface between the low-voltage control signals and the high-voltage motor driving circuitry. They provide electrical isolation, ensuring that high-voltage spikes in the driving circuit do not damage the microcontroller. When a control signal is applied to the input of the optocoupler, its internal LED emits light, triggering the phototransistor on the output side. This, in turn, switches the corresponding MOSFET on or off, thereby controlling the current path through the motor.

By activating pairs of MOSFETs on opposite sides of the H-bridge (one P-type and one N-type), the direction of current through the motor can be reversed, which allows for clockwise and counterclockwise rotation. If both upper or both lower MOSFETs are turned off, the motor will stop. This configuration allows precise control over the motor's direction and braking behavior. The detailed circuit diagrams can be found in Figure 1 and Figure 2.

Metal detector

An inductor installed at the bottom of the robot generates a magnetic field when energized. When a coin passes near it, the magnetic field is disrupted, causing a measurable change in the inductor's circuit behavior—typically a shift in the output signal's frequency.

To detect this change, the microcontroller continuously monitors the digital signal from the sensing circuit. A timer is used to precisely measure the signal's period by resetting and starting at specific signal edges. The frequency is then calculated from this period, and a moving average filter is applied to reduce noise and improve stability.

The system compares the measured frequency to a reference frequency (recorded when no metal is present). If the difference exceeds a set threshold, a coin detection flag is triggered.

Once a coin is detected, the robot executes a pickup sequence: the servo-controlled arm lowers, grabs the coin, lifts it, and returns to its default position. The robot then resumes searching until it collects the specified number of coins.

PWM (Pulse Width Modulation) Design

To regulate the motor's speed, a Pulse Width Modulation (PWM) signal is applied. PWM controls the motor by quickly turning the power supply on and off at a constant frequency. The duty cycle—the ratio of "on" time to the total cycle time—determines the average voltage supplied to the motor.

In the code, the microcontroller’s timer peripheral generates the PWM signal. The timer is set up to switch an output pin on and off at a specific frequency, while a compare value adjusts the duty cycle. For instance, a 70% duty cycle means the motor is powered for 70% of each cycle and idle for the remaining 30%.

By increasing the duty cycle, the motor receives a higher average voltage, speeding it up. Conversely, lowering the duty cycle reduces the voltage, slowing the motor down. This approach allows smooth and efficient speed control without altering the actual supply voltage.

Bluetooth

We use the HC-05 as our Bluetooth module and connect its RX and TX pins directly to the corresponding RX and TX pins of the JDY-40 module. This setup enables both modules to share the same data transmission functionality throughout the entire process. By utilizing the same UART2 interface for communication, our team ensures that both the JDY-40 and HC-05 modules transmit and receive data in a consistent and synchronized manner. This approach allows us to avoid using another UART function for bluetooth data transmission.

3.7 Solution Assessment

Remote Controller

Using push buttons to switch between different modes, we were able to seamlessly transition among four modes: manual picking, auto-picking, obstacle avoidance, and battery voltage display. In both manual and auto-picking modes, we employed the JDY-40 module to communicate with the robotic car, sending and receiving messages as needed. Specifically, in manual mode, we continuously send messages to direct the car in the desired direction. Additionally, when we press a push button to switch states, specific messages are transmitted to the robotic car. In auto mode, to accurately determine the detected frequency, the remote controller receives frequency data from the robotic car. For obstacle avoidance mode, we use the JDY-40 to activate this state, after which the HC-05 Bluetooth module on the robotic car takes over data transmission.

Robotic Car

Upon receiving various messages from the remote controller, the robotic car adjusts its behavior accordingly. In manual picking mode, it changes its speed and direction based on the received instructions. In auto-picking mode, when a coin is detected, the car sends the detected frequency to the remote controller. To ensure accurate communication, we transmit the data multiple times to confirm that the remote controller receives the precise frequency. Then, in obstacle avoidance mode, after receiving messages via the JDY-40, we disable the JDY-40 for data transmission between the remote controller and the robotic car and enable the HC-05 for Bluetooth communication with our computer, which is used to plot the car's path using Python.

4. Life-Long Learning

Throughout the course of this project, our team acquired several essential technical skills that were directly applied to the system's development and performance. These included generating PWM signals for motor speed control, implementing ADC techniques for precise sensor-based speed adjustments, and utilizing UART communication through the JDY-40 modules. Moreover, throughout the development process, we faced numerous challenges, especially during debugging due to the complexity of the system. This experience emphasized the importance of modular design in both hardware and software. By breaking the system into smaller, manageable components, debugging became more efficient and organized. This approach significantly enhanced our ability to identify and resolve issues during development.

During the project, we identified PWM control as a key knowledge gap—realizing that a deeper understanding at an earlier stage would have significantly improved our development efficiency and troubleshooting process.

This project also demonstrated how foundational knowledge from the curriculum supported our design process. CPSC 259 provided a solid background in C programming, which was crucial for developing and debugging embedded code efficiently. Additionally, ELEC 201 played a vital role by equipping us with practical understanding of MOSFETs, diodes, and capacitors, which directly supported our implementation of motor control circuits and the design of the metal detector subsystem.

5. Conclusion

In conclusion, Our project successfully met the design goals by integrating a robotic car using the EFM8 microcontroller with a remote controller based on STM32. We implemented multiple modes—including manual picking, auto-picking, obstacle avoidance, and battery monitoring—through reliable communication between the two systems.

The car's code uses Timer5 ISR interrupts to control PWM for motors and servos, while the remote controller allows mode switching and parameter adjustments via push buttons and an LCD. Wireless modules ensured continuous message exchange between the car and controller. The JDY-40 module handled communication in manual and auto-picking modes, while the HC-05 module enabled obstacle avoidance data to be sent to a computer for path plotting using Python. This setup allowed dynamic response to both user inputs and environmental changes.

Throughout the project, we encountered challenges such as synchronizing the servo and motor controls within the Timer5 interrupt routine, avoiding interference between multiple wireless modules, and ensuring reliable data transmission under different operating conditions. Iterative testing and debugging allowed us to refine our design and address these issues effectively.

This project took approximately 150 hours to complete. Around 10 hours were dedicated to preliminary research and information gathering, followed by 20 hours spent on circuit design and hardware setup. Implementing the core functionality required about 90 hours, and an additional 30 hours were used to integrate bonus features such as obstacle avoidance.

This project has significantly enhanced our skills in embedded system programming and real-time control, while also strengthening our abilities in teamwork and problem-solving.

6. Reference

[1] J. Calvino-Fraga, *ELEC291_Project_2_2025.pdf*, University of British Columbia, 2025

[2] J. Calvino-Fraga, *EFM8_JDY40_test.zip*, University of British Columbia, 2025

[3] J. Calvino-Fraga, *STM32L051_JDY40_Test.zip*, University of British Columbia, 2025

[4] J. Calvino-Fraga, *Robot_assembly.pdf*, University of British Columbia, 2025

[5] J. Calvino-Fraga, *Robot_assembly.pdf*, University of British Columbia, 2025

[6] J. Calvino-Fraga,*Making_the_electromagnet.pdf*, University of British Columbia, 2025

[7] J. Calvino-Fraga, *JDY40_Connector.pdf*, University of British Columbia, 2025

7. Bibliography

[1] Cytron Technologies Sdn. Bhd, Product User's Manual – HCSR04 Ultrasonic Sensor, May 2013.

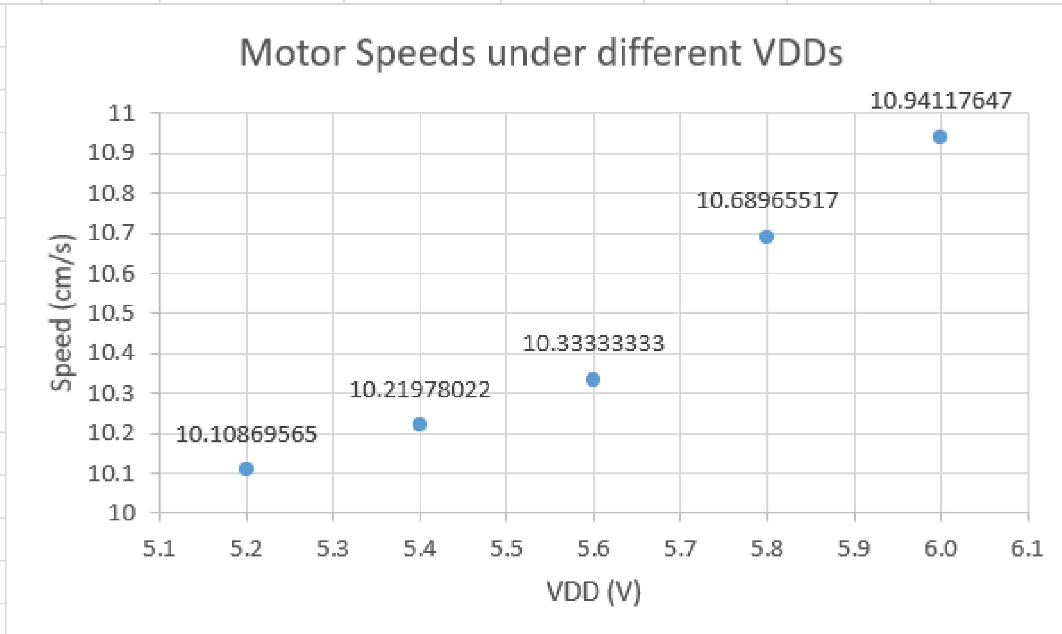
[2] www.banggood.com,Banggood HC-05 User Manual.

[3] https://picaxe.com/docs/spe033.pdf, DFPLayer Mini.

8. Appendices

8.1 Motor Speed Calculations

Length (cm)	VDD (V)	Time (s)	Speed (cm/s)		
93	5.2	9.2	10.1086957	Avg Speed:	10.45853
93	5.4	9.1	10.2197802		
93	5.6	9.0	10.3333333		
93	5.8	8.7	10.6896552		
93	6.0	8.5	10.9411765		



8.2 Source Code for Robotic Car

```
#include <EFM8LB1.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

volatile unsigned int pwm_reload;
volatile unsigned char pwm_state=0;
volatile unsigned char count20ms;
volatile unsigned char motor_flag=0;
volatile unsigned char servo_up=50, servo_down=150;
volatile unsigned int servo_counter = 0, isr_count = 0;
volatile unsigned int action_flag = 1;
volatile unsigned int threshold;
volatile unsigned int motor_counter = 0;
float pulse_width;
xdata float period = 0.0;
xdata float period_100 = 0.0;
xdata float period_sum = 0.0;
volatile unsigned int z;
unsigned int bluetooth_flag;

//xdata float pulse_width;
volatile unsigned int distance = 0;

#define PWMOUT_left P2_1
#define PWMOUT_left_inv P2_2
#define PWMOUT_right P2_5
#define PWMOUT_right_inv P2_6
//servo arm
```

```
#define SERVO_up P1_5
#define SERVO_down P1_7
#define EMAGNET  P1_4
#define VDD 3.3035

#define SYSCLK 72000000L // SYSCLK frequency in Hz
#define BAUDRATE 115200L
#define SARCLK 18000000L
//#define RELOAD_10MS (0x10000L-(SYSCLK/(12L*100L)))
#define RELOAD_10us (0x10000L-(SYSCLK/(12L*100000L))) // 10us rate
#define RELOAD_50us (0x10000L - (SYSCLK/(12L*20000L)))
#define SARCLK 18000000L
#define MOTOR_PWM_PERIOD_TICKS 20
//auto avd
#define TRIG_PIN 0x01 //1.0
#define ECHO_PIN 0x02 //1.1
#define DF_PLAY_PIN 0x08 //1.3
#define DISTANCE_THRESHOLD 5  // Trigger playback when within this distance
#define MAX_DISTANCE 50        // Reasonable maximum distance (cm)
#define MAX_TIMEOUT 30000      // Maximum timeout (microseconds)

// // DFPlayer
// #define DF_START_BYTE 0x7E
// #define DF_VERSION    0xFF
// #define DF_LENGTH     0x06
// #define DF_FEEDBACK   0x00
// #define DF_END_BYTE   0xEF
// #define DF_PLAY       0x03
// #define DF_SET_VOL    0x06
// #define DF_RESET      0x0C
// #define DF_PAUSE      0x0E
// #define DF_RESUME     0x0F
// #define DF_STOP       0x16

//idata char buff[20];
xdata char buff[20];
unsigned char overflow_count = 0;

char _c51_external_startup (void)
{
	// Disable Watchdog with key sequence
	SFRPAGE = 0x00;
	WDTCN = 0xDE; //First key
	WDTCN = 0xAD; //Second key

	VDM0CN |= 0x80;
	RSTSRC = 0x02;

	#if (SYSCLK == 48000000L)
		SFRPAGE = 0x10;
		PFE0CN  = 0x10; // SYSCLK < 50 MHz.
		SFRPAGE = 0x00;
	#elif (SYSCLK == 72000000L)
		SFRPAGE = 0x10;
```

```
        PFE0CN  = 0x20; // SYSCLK < 75 MHz.
        SFRPAGE = 0x00;
    #endif

    #if (SYSCLK == 12250000L)
        CLKSEL = 0x10;
        CLKSEL = 0x10;
        while ((CLKSEL & 0x80) == 0);
    #elif (SYSCLK == 24500000L)
        CLKSEL = 0x00;
        CLKSEL = 0x00;
        while ((CLKSEL & 0x80) == 0);
    #elif (SYSCLK == 48000000L)
        // Before setting clock to 48 MHz, must transition to 24.5 MHz first
        CLKSEL = 0x00;
        CLKSEL = 0x00;
        while ((CLKSEL & 0x80) == 0);
        CLKSEL = 0x07;
        CLKSEL = 0x07;
        while ((CLKSEL & 0x80) == 0);
    #elif (SYSCLK == 72000000L)
        // Before setting clock to 72 MHz, must transition to 24.5 MHz first
        CLKSEL = 0x00;
        CLKSEL = 0x00;
        while ((CLKSEL & 0x80) == 0);
        CLKSEL = 0x03;
        CLKSEL = 0x03;
        while ((CLKSEL & 0x80) == 0);
    #else
        #error SYSCLK must be either 12250000L, 24500000L, 48000000L, or
72000000L
    #endif

    // Configure the pins used for square output

    P0MDOUT |= 0x10; // Enable UART0 TX as push-pull output
    P2MDOUT |= 0X01;
    P0MDOUT |= 0x11; // Enable UART0 TX (P0.4) and UART1 TX (P0.0) as push-pull
outputs
    // P0SKIP |= 0b_0000_0001;
    // P0MDOUT |= 0b_0000_0010;
    XBR0     = 0x01; // Enable UART0 on P0.4(TX) and P0.5(RX)
    XBR1     = 0X10; // Enable T0 on P0.0
    XBR1     = 0X00; // Enable T0 on P0.0
    XBR2     = 0x40; // Enable crossbar and weak pull-ups
    XBR2     = 0x41;
    P2MDOUT |= 0x0F;
    //auto avd
    P1MDOUT |= TRIG_PIN | DF_PLAY_PIN; // Set TRIG and DF_PLAY to push-pull output
    P1MDOUT &= ~ECHO_PIN; // Ensure ECHO is in input mode
    P1 |= DF_PLAY_PIN;

    // Configure Uart 0
    #if (((SYSCLK/BAUDRATE)/(2L*12L))>0xFFL)
```

```
              #error Timer 0 reload value is incorrect because
(SYSCLK/BAUDRATE)/(2L*12L) > 0xFF
       #endif
       SCON0 = 0x10;
       TH1 = 0x100-((SYSCLK/BAUDRATE)/(2L*12L));
       TL1 = TH1;      // Init Timer1
       TMOD &= ~0xf0;  // TMOD: timer 1 in 8-bit auto-reload
       TMOD |=  0x20;
       TR1 = 1; // START Timer1
       TI = 1;  // Indicate TX0 ready

       P2_0=1; // 'set' pin to 1 is normal operation mode.

       //return 0;

       P2MDOUT |= 0x0F;
       P1MDOUT |= 0xC0;


    PWMOUT_left = 1;
    PWMOUT_left_inv = 1;
    PWMOUT_right = 1;
    PWMOUT_right_inv = 1;

       #if (((SYSCLK/BAUDRATE)/(2L*12L))>0xFFL)
              #error Timer 0 reload value is incorrect because
(SYSCLK/BAUDRATE)/(2L*12L) > 0xFF
       #endif
       // Configure Uart 0
       SCON0 = 0x10;
       CKCON0 |= 0b_0000_0000 ; // Timer 1 uses the system clock divided by 12.
       TH1 = 0x100-((SYSCLK/BAUDRATE)/(2L*12L));
       TL1 = TH1;      // Init Timer1
       TMOD &= ~0xf0;  // TMOD: timer 1 in 8-bit auto-reload
       TMOD |=  0x20;
       TR1 = 1; // START Timer1
       TI = 1;  // Indicate TX0 ready

       // Initialize timer 5 for periodic interrupts
       SFRPAGE=0x10;
       TMR5CN0=0x00;   // Stop Timer5; Clear TF5; WARNING: lives in SFR page 0x10
       //CKCON1|=0b_0000_0100; // Timer 5 uses the system clock
       pwm_reload=0x10000L-(SYSCLK*1.5e-3)/12.0; // 1.5 miliseconds pulse is the
center of the servo
       TMR5=0xffff;   // Set to reload immediately
       EIE2|=0b_0000_1000; // Enable Timer5 interrupts
       TR5=1;         // Start Timer5 (TMR5CN0 is bit addressable)

       EA=1;

       SFRPAGE=0x00;

       return 0;
}
```

```
void Timer5_ISR (void) interrupt INTERRUPT_TIMER5
{

        // Clear Timer5 interrupt flag
       // Since the maximum time we can achieve with this timer in the
       // configuration above is about 10ms, implement a simple state
       // machine to produce the required 20ms period.
       //servo_counter ++;
       if(action_flag==0){
              SFRPAGE=0x10;
              TF5H = 0;
              TMR5RL=RELOAD_10us;
              servo_counter++;
              if(servo_counter == 2000){
                     servo_counter=0;
              }
              if(servo_up>=servo_counter){
                     SERVO_up=1;
              }else{
                     SERVO_up=0;
              }
              if(servo_down>=servo_counter){
                     SERVO_down=1;
              }else{
                     SERVO_down=0;
              }
       }

       else if(action_flag==1){
              SFRPAGE=0x10;
              TF5H = 0;
              TMR5RL=RELOAD_50us;
              motor_counter++;
        if (motor_counter >= MOTOR_PWM_PERIOD_TICKS) {
            motor_counter = 0;
        }
        threshold =  (unsigned int)(pulse_width * MOTOR_PWM_PERIOD_TICKS);
        if (motor_counter < threshold) {
            switch (motor_flag) {
                case 1: // forward
                    PWMOUT_left = 0;
                    PWMOUT_right = 0;
                    PWMOUT_left_inv = 1;
                    PWMOUT_right_inv = 1;
                    break;
                case 2: // back
                    PWMOUT_left = 1;
                    PWMOUT_right = 1;
                    PWMOUT_left_inv = 0;
                    PWMOUT_right_inv = 0;
                    break;
                case 3: // right
                    PWMOUT_left = 0;
```

```
            PWMOUT_right = 1;
            PWMOUT_left_inv = 1;
            PWMOUT_right_inv = 0;
            break;
        case 4: // left
            PWMOUT_left = 1;
            PWMOUT_right = 0;
            PWMOUT_left_inv = 0;
            PWMOUT_right_inv = 1;
            break;
        case 5: // stop
                    PWMOUT_left = 1;
                    PWMOUT_right = 1;
                    PWMOUT_left_inv = 1;
                    PWMOUT_right_inv = 1;
                    break;
                case 6: //right_forward
                    PWMOUT_left = 0;
                    PWMOUT_right = 0;
                    PWMOUT_left_inv = 1;
                    PWMOUT_right_inv = 0;
                    break;
                case 7: //left_forward
                    PWMOUT_left = 0;
                    PWMOUT_right = 0;
                    PWMOUT_left_inv = 0;
                    PWMOUT_right_inv = 1;
                    break;
                case 8: //left_back
                    PWMOUT_left = 0;
                    PWMOUT_right = 1;
                    PWMOUT_left_inv = 0;
                    PWMOUT_right_inv = 0;
                    break;
                case 9: //right_back
                    PWMOUT_left = 1;
                    PWMOUT_right = 0;
                    PWMOUT_left_inv = 0;
                    PWMOUT_right_inv = 0;
                    break;
        default:
            PWMOUT_left = 1;
            PWMOUT_right = 1;
            PWMOUT_left_inv = 1;
            PWMOUT_right_inv = 1;
            break;
    }
} else {
    PWMOUT_left = 1;
    PWMOUT_right = 1;
    PWMOUT_left_inv = 1;
    PWMOUT_right_inv = 1;
}
```

```
        }
        if(isr_count  == 1000){
                isr_count = 0;
                period_sum = 0.0;
                for(z = 0; z < 10; z++){
                        TL0=0;
                        TH0=0;
                        overflow_count=0;
                        while(P0_2!=0); // Wait for the signal to be zero
                        while(P0_2!=1); // Wait for the signal to be one
                        TR0=1; // Start the timer
                        while(P0_2!=0) // Wait for the signal to be zero
                        {
                                if(TF0==1) // Did the 16-bit timer overflow?
                                {
                                        TF0=0;
                                        overflow_count++;
                                }
                        }
                        while(P0_2!=1) // Wait for the signal to be one
                        {
                                if(TF0==1) // Did the 16-bit timer overflow?
                                {
                                        TF0=0;
                                        overflow_count++;
                                }
                        }
                        TR0=0; // Stop timer 0, the 24-bit number
[overflow_count-TH0-TL0] has the period!
                        period=(overflow_count*65536.0+TH0*256.0+TL0)*(12.0/SYSCLK);
                        period_sum = period + period_sum;
                }
                period_100 = period_sum / 10.0;
        }
        isr_count++;
}
void InitADC (void)
{
        SFRPAGE = 0x00;
        ADEN=0; // Disable ADC

        ADC0CN1=
                (0x2 << 6) | // 0x0: 10-bit, 0x1: 12-bit, 0x2: 14-bit
          (0x0 << 3) | // 0x0: No shift. 0x1: Shift right 1 bit. 0x2: Shift right 2
bits. 0x3: Shift right 3 bits.
                (0x0 << 0) ; // Accumulate n conversions: 0x0: 1, 0x1:4, 0x2:8, 0x3:16,
0x4:32

        ADC0CF0=
            ((SYSCLK/SARCLK) << 3) | // SAR Clock Divider. Max is 18MHz. Fsarclk =
(Fadcclk) / (ADSC + 1)
                (0x0 << 2); // 0:SYSCLK ADCCLK = SYSCLK. 1:HFOSC0 ADCCLK = HFOSC0.

        ADC0CF1=
```

```
		(0 << 7)   | // 0: Disable low power mode. 1: Enable low power mode.
		(0x1E << 0); // Conversion Tracking Time. Tadtk = ADTK / (Fsarclk)

	ADC0CN0 =
		(0x0 << 7) | // ADEN. 0: Disable ADC0. 1: Enable ADC0.
		(0x0 << 6) | // IPOEN. 0: Keep ADC powered on when ADEN is 1. 1: Power
down when ADC is idle.
		(0x0 << 5) | // ADINT. Set by hardware upon completion of a data
conversion. Must be cleared by firmware.
		(0x0 << 4) | // ADBUSY. Writing 1 to this bit initiates an ADC
conversion when ADCM = 000. This bit should not be polled to indicate when a
conversion is complete. Instead, the ADINT bit should be used when polling for
conversion completion.
		(0x0 << 3) | // ADWINT. Set by hardware when the contents of
ADC0H:ADC0L fall within the window specified by ADC0GTH:ADC0GTL and ADC0LTH:ADC0LTL.
Can trigger an interrupt. Must be cleared by firmware.
		(0x0 << 2) | // ADGN (Gain Control). 0x0: PGA gain=1. 0x1: PGA
gain=0.75. 0x2: PGA gain=0.5. 0x3: PGA gain=0.25.
		(0x0 << 0) ; // TEMPE. 0: Disable the Temperature Sensor. 1: Enable the
Temperature Sensor.

	ADC0CF2=
		(0x0 << 7) | // GNDSL. 0: reference is the GND pin. 1: reference is the
AGND pin.
		(0x1 << 5) | // REFSL. 0x0: VREF pin (external or on-chip). 0x1: VDD
pin. 0x2: 1.8V. 0x3: internal voltage reference.
		(0x1F << 0); // ADPWR. Power Up Delay Time. Tpwrtime = ((4 * (ADPWR +
1)) + 2) / (Fadcclk)

	ADC0CN2 =
		(0x0 << 7) | // PACEN. 0x0: The ADC accumulator is over-written.  0x1:
The ADC accumulator adds to results.
		(0x0 << 0) ; // ADCM. 0x0: ADBUSY, 0x1: TIMER0, 0x2: TIMER2, 0x3:
TIMER3, 0x4: CNVSTR, 0x5: CEX5, 0x6: TIMER4, 0x7: TIMER5, 0x8: CLU0, 0x9: CLU1, 0xA:
CLU2, 0xB: CLU3

	ADEN=1; // Enable ADC
}

void InitPinADC (unsigned char portno, unsigned char pinno)
{
	unsigned char mask;

	mask=1<<pinno;

	SFRPAGE = 0x20;
	switch (portno)
	{
		case 0:
			P0MDIN &= (~mask); // Set pin as analog input
			P0SKIP |= mask; // Skip Crossbar decoding for this pin
		break;
		case 1:
			P1MDIN &= (~mask); // Set pin as analog input
```

```
                P1SKIP |= mask; // Skip Crossbar decoding for this pin
            break;
            case 2:
                P2MDIN &= (~mask); // Set pin as analog input
                P2SKIP |= mask; // Skip Crossbar decoding for this pin
            break;
            default:
            break;
        }
        SFRPAGE = 0x00;
}
unsigned int ADC_at_Pin(unsigned char pin)
{
        ADC0MX = pin;   // Select input from pin
        ADINT = 0;
        ADBUSY = 1;     // Convert voltage at the pin
        while (!ADINT); // Wait for conversion to complete
        return (ADC0);
}

float Volts_at_Pin(unsigned char pin)
{
         return ((ADC_at_Pin(pin)*VDD)/0b_0011_1111_1111_1111);
}
void TIMER0_Init(void)
{
        TMOD&=0b_1111_0000; // Set the bits of Timer/Counter 0 to zero
        TMOD|=0b_0000_0001; // Timer/Counter 0 used as a 16-bit timer
        TR0=0; // Stop Timer/Counter 0
}
// Uses Timer3 to delay <us> micro-seconds.
void Timer3us(unsigned char us)
{
        unsigned char i;                // usec counter

        // The input for Timer 3 is selected as SYSCLK by setting T3ML (bit 6) of
CKCON0:
        CKCON0|=0b_0100_0000;

        TMR3RL = (-(SYSCLK)/1000000L); // Set Timer3 to overflow in 1us.
        TMR3 = TMR3RL;                 // Initialize Timer3 for first overflow

        TMR3CN0 = 0x04;                 // Sart Timer3 and clear overflow flag
        for (i = 0; i < us; i++)       // Count <us> overflows
        {
                while (!(TMR3CN0 & 0x80));  // Wait for overflow
                TMR3CN0 &= ~(0x80);         // Clear overflow indicator
        }
        TMR3CN0 = 0 ;                   // Stop Timer3 and clear overflow flag
}

void waitms (unsigned int ms)
{
        unsigned int j;
```

```
        for(j=ms; j!=0; j--)
        {
                Timer3us(249);
                Timer3us(249);
                Timer3us(249);
                Timer3us(250);
        }
}
//slave
void UART1_Init (unsigned long baudrate)
{
    SFRPAGE = 0x20;
        SMOD1 = 0x0C; // no parity, 8 data bits, 1 stop bit
        SCON1 = 0x10;
        SBCON1 =0x00;   // disable baud rate generator
        SBRL1 = 0x10000L-((SYSCLK/baudrate)/(12L*2L));
        TI1 = 1; // indicate ready for TX
        SBCON1 |= 0x40;   // enable baud rate generator
        SFRPAGE = 0x00;
}

void putchar1 (char c)
{
    SFRPAGE = 0x20;
        while (!TI1);
        TI1=0;
        SBUF1 = c;
        SFRPAGE = 0x00;
}

void sendstr1 (char * s)
{
        while(*s)
        {
                putchar1(*s);
                s++;
        }
}

char getchar1 (void)
{
        char c;
    SFRPAGE = 0x20;
        while (!RI1);
        RI1=0;
        // Clear Overrun and Parity error flags
        SCON1&=0b_0011_1111;
        c = SBUF1;
        SFRPAGE = 0x00;
        return (c);
}

char getchar1_with_timeout (void)
{
```

```
	char c;
	unsigned int timeout;
    SFRPAGE = 0x20;
    timeout=0;
	while (!RI1)
	{
		SFRPAGE = 0x00;
		Timer3us(20);
		SFRPAGE = 0x20;
		timeout++;
		if(timeout==25000)
		{
			SFRPAGE = 0x00;
			return ('\n'); // Timeout after half second
		}
	}
	RI1=0;
	// Clear Overrun and Parity error flags
	SCON1&=0b_0011_1111;
	c = SBUF1;
	SFRPAGE = 0x00;
	return (c);
}

void getstr1 (char * s, unsigned char n)
{
	char c;
	unsigned char cnt;

	cnt=0;
	while(1)
	{
		c=getchar1_with_timeout();
		if(c=='\n')
		{
			*s=0;
			return;
		}

		if (cnt<n)
		{
			cnt++;
			*s=c;
			s++;
		}
		else
		{
			*s=0;
			return;
		}
	}
}

// RXU1 returns '1' if there is a byte available in the receive buffer of UART1
```

```
bit RXU1 (void)
{
       bit mybit;
    SFRPAGE = 0x20;
       mybit=RI1;
       SFRPAGE = 0x00;
       return mybit;
}

void waitms_or_RI1 (unsigned int ms)
{
       unsigned int j;
       unsigned char k;
       for(j=0; j<ms; j++)
       {
              for (k=0; k<4; k++)
              {
                     if(RXU1()) return;
                     Timer3us(250);
              }
       }
}

void SendATCommand (char * s)
{
       printf("Command: %s", s);
       P2_0=0; // 'set' pin to 0 is 'AT' mode.
       waitms(5);
       sendstr1(s);
       getstr1(buff, sizeof(buff)-1);
       waitms(10);
       P2_0=1; // 'set' pin to 1 is normal operation mode.
       printf("Response: %s\r\n", buff);
}

void ReceptionOff (void)
{

       P2_0=0; // 'set' pin to 0 is 'AT' mode.
       waitms(10);
       sendstr1("AT+DVID0000\r\n"); // Some unused id, so that we get nothing in
RXD1.
       waitms(10);
       // Clear Overrun and Parity error flags
       SCON1&=0b_0011_1111;
       P2_0=1; // 'set' pin to 1 is normal operation mode.
}
//auto avd
void play_sound_via_gpio(void)
{
    P1 &= ~DF_PLAY_PIN;
    waitms(200);
    P1 |= DF_PLAY_PIN;
    //printf("* Playing sound *\n");
```

```
}
unsigned int measure_distance(void)
{
    unsigned int duration = 0;
    unsigned int distance;

    // Send trigger pulse
    P1 &= ~TRIG_PIN;    // Ensure TRIG is initially low
    Timer3us(2);        // Short delay
    P1 |= TRIG_PIN;     // Set TRIG high
    Timer3us(15);       // Pulse for at least 10 microseconds 3
    P1 &= ~TRIG_PIN;    // Set TRIG low

    // Wait for ECHO rising edge with timeout
    duration = 0;
    while ((P1 & ECHO_PIN) == 0)
    {
        Timer3us(10);
        duration += 10;
        if (duration > MAX_TIMEOUT)
        {
           // printf("Error: Rising edge timeout\n");
            return 0xFFFF;   // Return error value
        }
    }

    // Reset timer
    duration = 0;

    // Measure the duration of ECHO high pulse
    while ((P1 & ECHO_PIN) != 0)
    {
        Timer3us(10);
        duration += 10;
        if (duration > MAX_TIMEOUT)
        {
            //printf("Error: Falling edge timeout\n");
            return 0xFFFF;   // Return error value
        }
    }

    // Calculate distance (cm) - Speed of sound = 340 m/s, divide by 2 for round
trip

    distance = duration / 59;

    // Filter out abnormal values
    if (distance > MAX_DISTANCE)
    {
        //printf("Error: Distance too large (%u cm)\n", distance);
        return 0xFFFF;  // Return error value
    }

    //printf("Raw distance: %u cm (pulse duration: %u us)\n", distance, duration);
```

```
    return distance;
}
unsigned int filtered_measure_distance(void)
{
#define FILTER_SIZE 3
    unsigned int readings[FILTER_SIZE];
    unsigned int sum = 0, valid_count = 0;
    unsigned int i;

    //printf("\n--- Starting new measurement ---\n");

    // Get multiple readings
    for (i = 0; i < FILTER_SIZE; i++)
    {
        //printf("Sample %u: ", i + 1);
        readings[i] = measure_distance();

        if (readings[i] != 0xFFFF && readings[i] <= MAX_DISTANCE)
        {
            sum += readings[i];
            valid_count++;
        }
        waitms(20);  // Wait 20ms before the next measurement
    }

    // If there are valid readings, return the average
    if (valid_count > 0)
    {
        unsigned int avg = sum / valid_count;
       // printf("Average distance: %u cm (from %u valid readings)\n", avg,
valid_count);
        return avg;
    }
    else
    {
        //printf("No valid readings\n");
        return 0xFFFF;
    }
}

////////////
void main (void)
{
    //float pulse_width;
      xdata int get_voltagex = -1500, get_voltagey = -1500;
      int datachanged = 0; //data didn't change
      ///arm
      unsigned char j, k;
    ////slave
      unsigned int cnt=0;
      xdata char auto_flag = 0;
      int skip_servo = 0;
      int coin_count = 0;
    char c;
```

```
        ///detecter
        //float period;
        xdata float v[4];
        float vsum;
        int coinflag = 0;
    int perimeterflag = 0;
        int i;
        xdata float freq;
        xdata float  freq_buffer[10] = {0};
        int buf_index = 0;
        int freq_flag = 1;
        float xdata freq_base = 0;
        xdata float speed;
        xdata int send_message_count = 0;
        xdata long int freq_int;
        xdata float m_voltage = 0;
        long int m_voltage_int = 0;
        P1_4 = 0;
        //bluetooth



        TIMER0_Init();
        InitPinADC(0, 6); // Configure P0.6 as analog input
        InitPinADC(0, 7); // Configure P0.7 as analog input
        InitPinADC(2, 4); // Configure P2.4 as analog input
        InitADC();

        UART1_Init(9600);
        ReceptionOff();
        // DFPlayer_Init(); //new way for dfplayer

        // To check configuration
        SendATCommand("AT+VER\r\n");
        SendATCommand("AT+BAUD\r\n");
        SendATCommand("AT+RFID\r\n");
        SendATCommand("AT+DVID\r\n");
        SendATCommand("AT+RFC\r\n");
        SendATCommand("AT+POWE\r\n");
        SendATCommand("AT+CLSS\r\n");
        // We should select an unique device ID.  The device ID can be a hex
        // number from 0x0000 to 0xFFFF.  In this case is set to 0xABBA
        SendATCommand("AT+DVIDCBCB\r\n");

        cnt=0;
        action_flag=0;
        servo_up=50;
        waitms(500);
        servo_down=150;
        waitms(500);
    count20ms = 0; // Count20ms is an atomic variable, so no problem sharing with
timer 5 ISR
    // In a HS-422 servo a pulse width between 0.6 to 2.4 ms gives about 180 deg
    // of rotation range.
```

```
        motor_flag = 0;
        servo_counter=0;
        waitms(20);
        while(1)
        {
                servo_counter=0;
                //printf("while pass\n");
                if(RXU1()) // Something has arrived
                {
                        c=getchar1();

                        if(c=='!') // Master is sending message
                        {
                                getstr1(buff, sizeof(buff)-1);
                                if(strlen(buff)==7)
                                {
                                        printf("%s\r\n", buff);
                                        sscanf(buff, "%d %d", &get_voltagex, &get_voltagey)
== 2;
                                }
                                else
                                {
                                        printf("*** BAD MESSAGE ***: %s\r\n", buff);
                                }
                        }
                        else if(c=='@') // Master wants slave data
                        {
                                sprintf(buff, "%05u\n", cnt);
                                //cnt++;
                                waitms(5); // The radio seems to need this delay...
                                //sendstr1(buff);
                        }

                }

                //printf("dataflag= %d\n", datachanged);
                //printf("vx= %f, vy= %f\n", get_voltagex, get_voltagey);
                if(get_voltagex < -1000 || get_voltagey < -1000){
                        auto_flag = 0;
                        motor_flag = 5;

                }

                else if(get_voltagex == -20 && get_voltagey == -20 ){
                        printf("autoflag = %d\n", auto_flag);
                         if(auto_flag == 1){
                                printf("enter continue\n");
                                continue;
                        }
                //start auto
                        auto_flag = 1;
                        freq_flag = 1;
                        while(coin_count < 23)
                        {
```

```
                        printf("in while 20\n");
                        // first detect the ADC output
                        // Send the period to the serial port
                        freq = 1/period_100;
                        freq_int = (long int) freq;
                        if (freq_flag)
                        {
                                freq_flag = 0;
                                freq_base = freq;
                        }
                        printf("freq = %f, freq_base = %f", freq, freq_base);

                        if(freq <= freq_base + 350){
                                coinflag = 0;
                        }
                        else if(freq > freq_base + 350){
                                coinflag = 1;
                        }
                        v[0] = Volts_at_Pin(QFP32_MUX_P0_6);
                        v[2] = Volts_at_Pin(QFP32_MUX_P0_7);
                        vsum = v[0] + v[2];

                        if (vsum>=1.5)
                        {
                                printf ("Perimeter detected:v2.2=%10.8fV,
v2.4=%10.8fV, flag = 1\n",v[0], v[2]);
                                perimeterflag = 1;
                        }
                        else
                        {
                                // printf ("Perimeter not detected:v2.2=%10.8fV,
v2.4=%10.8fV, flag = 0\n",v[0], v[2]);
                                perimeterflag = 0;
                        }
                        if(perimeterflag == 0){
                                action_flag = 1;
                                motor_flag = 1;
                                waitms(20);
                                pulse_width=0.6;
                        }
                        else if(perimeterflag ==1){
                                while (vsum>1.5)// keep turning right until it
leaves perimeter
                                {
                                        action_flag = 1;
                                        motor_flag = 3;
                                        waitms(20);
                                        pulse_width=0.5;
                                        waitms(1200);
                                        v[0] = Volts_at_Pin(QFP32_MUX_P0_6);
                                        v[2] = Volts_at_Pin(QFP32_MUX_P0_7);
                                        vsum = v[0] + v[2];
                                }
```

```
				}
				if(coinflag == 1){
					while(send_message_count < 10){
						if(RXU1()) // Something has arrived
						{
							c=getchar1();

							if(c=='!') // Master is sending
message
							{
								getstr1(buff, sizeof(buff)-1);
								if(strlen(buff)==7)
								{
									printf("%s\r\n", buff);
									sscanf(buff, "%d %d",
&get_voltagex, &get_voltagey) == 2;
								}
								else
								{
									printf("*** BAD MESSAGE
***: %s\r\n", buff);
								}
							}
							else if(c=='@') // Master wants slave
data
							{
								sprintf(buff, "%ld\n",
freq_int);
								printf("send = %ld\n",
freq_int);
								waitms(20); // The radio seems
to need this delay...
								sendstr1(buff);
								waitms(20);
								sendstr1(buff);
								waitms(20);
								sendstr1(buff);
								waitms(20);
								sendstr1(buff);
								waitms(20);
								sendstr1(buff);
							}

						}
						send_message_count++;
					}
					send_message_count = 0;
					coin_count++;

					waitms(20);
					action_flag = 1;


					motor_flag =2; //go backwards for .5 seconds
```

```
waitms(10);
pulse_width=0.7;
waitms(330);
motor_flag =5;
waitms(50);
pulse_width=0;
waitms(50);
action_flag=0;
servo_counter=0;
waitms(20);
P1_4=0;

servo_up=50;
waitms(500);
servo_down=150;

waitms(500);
servo_up=50;
waitms(500);
servo_down=150;
waitms(2000);

printf("Servo has been initialized.\n");


for(j=50; j<=240; j+=5)
{
	servo_up = j;
	waitms(20);
}
//servo_up=240;
// waitms(100);
P1_4=1;
waitms(1800);

for(j=150; j<195; j+=5)
{
	servo_down = j;
	waitms(60);
}
waitms(500);

for(j=195; j>170; j-=5)
{
	servo_down = j;
	waitms(60);
}
waitms(1000);

//up: 240-55  185  down: 170-95  75

for(j=0;j<=37;j++){
	servo_up = (240-j*5);
	if(j>=22){
```

```
                        servo_down = (170-(j-21)*5);
                        waitms(5);
                }
                waitms(20);
            }


            waitms(1000);
            P1_4=0;
            waitms(500);
            printf("coin number = %d\n", coin_count);
            coinflag = 0;
        }
    }
    get_voltagex = -1500;
    get_voltagey = -1500;
    coin_count = 0;
    continue;
}


else if(get_voltagex == -10 && get_voltagey == -10) {
    if (skip_servo)
        continue;

    skip_servo = 1;

    printf("Asked to move servo, wheel has been stopped.\n");

    //printf("Servo has been initialized.\n");

    waitms(20);
    action_flag = 1;


            motor_flag =5;
            waitms(50);
            pulse_width=0;
            waitms(100);
            action_flag=0;
            servo_counter=0;
            waitms(20);
            P1_4=0;

            servo_up=50;
            waitms(500);
            servo_down=150;//
            waitms(500);
            for(j=150; j<110; j-=4)
            {
                servo_down = j;
                waitms(105);
            }
```

```
                                servo_up=50;
                                waitms(500);
                                for(j=110; j<155; j+=3)
                                {
                                        servo_down = j;
                                        waitms(100);
                                }

                                waitms(1000);

                                printf("Servo has been initialized.\n");


                                for(j=50; j<=240; j+=5)
                                {
                                        servo_up = j;
                                        waitms(60);
                                }
                                //servo_up=240;
                                // waitms(100);
                                P1_4=1;
                                waitms(1800);

                                for(j=155; j<191; j+=4)
                                {
                                        servo_down = j;
                                        waitms(80);
                                }
                                waitms(500);

                                for(j=191; j>170; j-=2)
                                {
                                        servo_down = j;
                                        waitms(80);
                                }
                                waitms(1000);

                                //up: 240-55  185  down: 170-95  75

                                for(j=0;j<=37;j++){
                                        servo_up = (240-j*5);
                                        if(j>=22){
                                                servo_down = (170-(j-21)*5);
                                                waitms(5);
                                        }
                                        waitms(20);
                                }
                                waitms(1000);
                                P1_4=0;
                                waitms(500);

                }
        else if(get_voltagex ==  -30 && get_voltagey == -30)
```

```
	{
		freq_flag = 1;
		printf("start auto avd\n");
		while(coin_count < 5)
		{
			action_flag = 1;
			motor_flag  = 1;
			pulse_width = 0.75;
			//send message to hc-05
			bluetooth_flag = 1;
			sprintf(buff, "%d\r\n", bluetooth_flag); // Construct a test
message
			waitms(5); // This may need adjustment depending on how busy is
the slave
			sendstr1(buff); // Send the test message

			distance = filtered_measure_distance();
			while(distance != 0xFFFF)
			{		//test if there is smth im front of the sensor
					//if smth is within the threshold distance
				while (distance < DISTANCE_THRESHOLD)
				{
					printf("detected\n");
					play_sound_via_gpio();
					action_flag = 1;
					motor_flag = 5;
					waitms(50);
					pulse_width=0; //stop the wheel
					//send message to hc-05
					// bluetooth_flag = 0; //stop
					// sprintf(buff, "%d\r\n", bluetooth_flag); //
Construct a test message
					// waitms(5); // This may need adjustment depending
on how busy is the slave
					// sendstr1(buff); // Send the test message
					printf("wheel stops\n");

					waitms(800);
					action_flag = 1;
					motor_flag = 3;
					waitms(20);
					pulse_width=0.75;
					waitms(800); //spin
					printf("wheel spins");
					//send message to hc-05
					bluetooth_flag = 3; //spin
					sprintf(buff, "%d\r\n", bluetooth_flag); //
Construct a test message
					waitms(5); // This may need adjustment depending on
how busy is the slave
					sendstr1(buff); // Send the test message
					distance = filtered_measure_distance();

				}
```

```
                        while (distance >= DISTANCE_THRESHOLD)
                        {

                                printf("in while 20\n");
                                // first detect the ADC output
                                // Send the period to the serial port
                                freq = 1/period_100;
                                if (freq_flag)
                                {
                                        freq_flag = 0;
                                        freq_base = freq;
                                }
                                printf("freq = %f, freq_base = %f", freq,
freq_base);

                                if(freq <= freq_base + 350){
                                        coinflag = 0;
                                }
                                else if(freq > freq_base + 350){
                                        coinflag = 1;

                                }
                                v[0] = Volts_at_Pin(QFP32_MUX_P0_6);
                                v[2] = Volts_at_Pin(QFP32_MUX_P0_7);
                                vsum = v[0] + v[2];

                                if (vsum>=1.5)
                                {
                                        printf ("Perimeter detected:v2.2=%10.8fV,
v2.4=%10.8fV, flag = 1\n",v[0], v[2]);
                                        perimeterflag = 1;
                                }
                                else
                                {
                                        // printf ("Perimeter not
detected:v2.2=%10.8fV, v2.4=%10.8fV, flag = 0\n",v[0], v[2]);
                                        perimeterflag = 0;
                                }
                                if(perimeterflag == 0){
                                        action_flag = 1;
                                        motor_flag = 1;
                                        waitms(20);
                                        pulse_width=0.75;
                                        //send message to hc-05
                                        bluetooth_flag = 1;
                                        sprintf(buff, "%d\r\n", bluetooth_flag); //
Construct a test message

                                        waitms(5); // This may need adjustment
depending on how busy is the slave
                                        sendstr1(buff); // Send the test message

                                }
                                else if(perimeterflag ==1){
                                        while (vsum>1.5)// keep turning right until
```

```
it leaves perimeter
                                        {
                                                action_flag = 1;
                                                motor_flag = 3;
                                                waitms(20);
                                                pulse_width=0.75;
                                                waitms(800);
                                                //send message to hc-05
                                                bluetooth_flag = 3;
                                                sprintf(buff, "%d\r\n",
bluetooth_flag); // Construct a test message
                                                waitms(5); // This may need adjustment
depending on how busy is the slave
                                                sendstr1(buff); // Send the test
message
                                                v[0] = Volts_at_Pin(QFP32_MUX_P0_6);
                                                v[2] = Volts_at_Pin(QFP32_MUX_P0_7);
                                                vsum = v[0] + v[2];
                                        }

                                }
                                if(coinflag == 1){

                                        waitms(20);

                                        waitms(20);
                                        action_flag = 1;
                                        //send message to hc-05
                                        bluetooth_flag = 4;//pick up coins
                                        sprintf(buff, "%d\r\n", bluetooth_flag); //
Construct a test message
                                        waitms(5); // This may need adjustment
depending on how busy is the slave
                                        sendstr1(buff); // Send the test message

                                        motor_flag =2; //go backwards for .5 seconds
                                        waitms(10);
                                pulse_width=0.7;
                                waitms(260);
                                        //send message to hc-05
                                        bluetooth_flag = 2;//back
                                        sprintf(buff, "%d\r\n", bluetooth_flag);
//mes
                                        waitms(5); // This may need adjustment
depend
                                        sendstr1(buff); // Send the test message
                                        waitms(295);
                                        motor_flag =5;
                                        waitms(50);
                                        pulse_width=0;
                                        waitms(500);
                                        //send message to hc-05
                                        // bluetooth_flag = 0;//stop
                                        // sprintf(buff, "%d\r\n", bluetooth_flag);
```

```
//
                                        // waitms(5); // This may need adjustment
                                               // sendstr1(buff); // Send the test
message
                                        action_flag=0;
                                        servo_counter=0;
                                        waitms(20);
                                        P1_4=0;
                                        servo_up=50;
                                waitms(500);
                                servo_down=150;//
                                waitms(500);
                                for(j=150; j<110; j-=4)
                                {
                                        servo_down = j;
                                        waitms(105);
                                }


                                servo_up=50;
                                waitms(500);
                                for(j=110; j<155; j+=3)
                                {
                                        servo_down = j;
                                        waitms(100);
                                }

                                waitms(1000);

                                printf("Servo has been initialized.\n");


                                for(j=50; j<=240; j+=5)
                                {
                                        servo_up = j;
                                        waitms(60);
                                }
                                //servo_up=240;
                                // waitms(100);
                                P1_4=1;
                                waitms(1800);

                                for(j=155; j<191; j+=4)
                                {
                                        servo_down = j;
                                        waitms(80);
                                }
                                waitms(500);

                                for(j=191; j>170; j-=2)
                                {
                                        servo_down = j;
                                        waitms(80);
                                }
```

```
                              waitms(1000);

                              //up: 240-55  185  down: 170-95  75

                              for(j=0;j<=37;j++){
                                    servo_up = (240-j*5);
                                    if(j>=22){
                                          servo_down = (170-(j-21)*5);
                                          waitms(5);
                                    }
                                    waitms(20);
                              }
                              waitms(1000);
                                    P1_4=0;
                                    waitms(500);
                                    printf("coin number = %d\n", coin_count);
                                    coinflag = 0;
                              }
                                    printf("go forward\n");
                                    action_flag = 1;
                                    motor_flag = 1;
                                    waitms(20);
                                    pulse_width=0.75;
                                    //send message to hc-05
                                    bluetooth_flag = 1;
                                    sprintf(buff, "%d\r\n", bluetooth_flag); //
                                    waitms(5); // This may need adjustment
depending

                                    sendstr1(buff); // Send the test message
                                    distance = filtered_measure_distance();
                        }
                  }
            }
                  coin_count = 0;
                  //send message to hc-05
                  bluetooth_flag = 0;//stop
                  sprintf(buff, "%d\r\n", bluetooth_flag); // Construct a test
message
                  waitms(5); // This may need adjustment depending on how busy is
the slave
                  sendstr1(buff); // Send the test message
      }
      else if(get_voltagex ==  -40 && get_voltagey == -40){
            v[1] = Volts_at_Pin(QFP32_MUX_P2_4);
            m_voltage = v[1];
            m_voltage = m_voltage/0.986*(0.986+2.983);
            m_voltage = m_voltage * 10000;
            m_voltage_int = (long int) m_voltage;
            if(RXU1()) // Something has arrived
            {
                  c=getchar1();

                  if(c=='!') // Master is sending message
                  {
```

```
                        getstr1(buff, sizeof(buff)-1);
                        if(strlen(buff)==7)
                        {
                                printf("%s\r\n", buff);
                        }
                        else
                        {
                                printf("*** BAD MESSAGE ***: %s\r\n", buff);
                        }
                }
                else if(c=='@') // Master wants slave data
                {
                        sprintf(buff, "%ld\n", m_voltage_int);
                        printf("send = %ld\n", m_voltage_int);
                        waitms(20); // The radio seems to need this delay...
                        sendstr1(buff);
                        waitms(20);
                        sendstr1(buff);
                        waitms(20);
                        sendstr1(buff);
                        waitms(20);
                        sendstr1(buff);
                        waitms(20);
                        sendstr1(buff);
                }

        }


}
        else {
                //printf("Asked to move wheel, servo cannot be moved. \n");
                action_flag=1;
                if((get_voltagex > 150) &&(get_voltagex < 180) &&
                (get_voltagey > 150) && (get_voltagey < 180)){

                        motor_flag = 5; //stop
                        pulse_width=0;
                        //printf("enter stop, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);
                }else if((get_voltagey > 180 && get_voltagey < 500)&&
(get_voltagex > 150 && get_voltagex < 180)){
                        speed=(float)(get_voltagey-180)/(350-180);
                        motor_flag = 1; //forward
                        pulse_width=0.75*speed;
                        printf("enter forward, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);
                }else if((get_voltagey < 150 && get_voltagey > -1)&&
(get_voltagex > 150 && get_voltagex < 180)){
                        speed=(float)(170-get_voltagey)/(170);
                        motor_flag = 2; //back
                        pulse_width=0.75*speed;

                        //printf("enter b, flag:%d,
```

```
pw:%.2f\n",motor_flag,pulse_width);

                    }else if((get_voltagex < 500 && get_voltagex > 180) &&
(get_voltagey > 150 && get_voltagey < 180)){
                            speed=(float)(get_voltagex-180)/(350-180);
                            motor_flag = 3; //right
                            pulse_width=0.75*speed;
                            //printf("enter r, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);

                    }else if((get_voltagex < 180 && get_voltagex > -1) &&
(get_voltagey > 150 && get_voltagey < 180)){
                            speed=(float)(170-get_voltagex)/(170);
                            motor_flag = 4; //left
                            pulse_width=0.75*speed;

                            //printf("enter l, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);

                    }else if((get_voltagex > 180 && get_voltagex < 500) &&
(get_voltagey > 180 && get_voltagey < 500)){
                            speed=(float)(get_voltagey-180)/(350-180);
                            motor_flag = 6;  //right forward
                            pulse_width=0.75*speed;

                            //printf("enter rf, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);

                    }else if((get_voltagex < 150 && get_voltagex > -1) &&
(get_voltagey > 180 && get_voltagey < 500)){
                            speed=(float)(get_voltagey-180)/(350-180);
                            motor_flag = 7; //left forward
                            pulse_width=0.75*speed;

                            //printf("enter lf, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);

                    }else if((get_voltagex > 180 && get_voltagex < 500)&&
(get_voltagey < 150 && get_voltagey > -1)){
                            speed=(float)(170-get_voltagey)/(170);
                            motor_flag = 9; //right back
                            pulse_width=0.75*speed;

                            //printf("enter rb, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);

                    }else if((get_voltagex < 150 && get_voltagex > -1) &&
(get_voltagey < 150 && get_voltagey > -1)){
                            speed=(float)(170-get_voltagey)/(170);
                            motor_flag = 8; //left back
                            pulse_width=0.75*speed;
                            //printf("enter lb, flag:%d,
pw:%.2f\n",motor_flag,pulse_width);
```

```
			}
				skip_servo = 0;
				auto_flag = 0;
			//printf("Motion of wheel has been set, waiting for next input.
\n");
		}


	}
}
```

8.3 Source Code for Remote Controller

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "../Common/Include/stm32l051xx.h"
#include "../Common/Include/serial.h"
#include "adc.h"
#include "lcd.h"
#include "UART2.h"

#define F_CPU 32000000L
#define SYSCLK 32000000L
#define DEF_F 15000L
//#define TICK_FREQ 2000L

volatile int Count = 0;

void ToggleLED(void)
{
	GPIOA->ODR ^= BIT8; // Toggle PA8
}

void TIM21_Handler(void)
{
	TIM21->SR &= ~BIT0; // clear update interrupt flag
	ToggleLED(); // toggle the state of the LED every half second
}

// LQFP32 pinout with the pins that can be analog inputs.  This code uses ADC_IN9.
//                  ----------
//           VDD -|1       32|- VSS
//          PC14 -|2       31|- BOOT0
//          PC15 -|3       30|- PB7
//          NRST -|4       29|- PB6
//          VDDA -|5       28|- PB5
// (ADC_IN0) PA0 -|6       27|- PB4
// (ADC_IN1) PA1 -|7       26|- PB3
// (ADC_IN2) PA2 -|8       25|- PA15 //connect to JDY40
// (ADC_IN3) PA3 -|9       24|- PA14 //connect to JDY40
// (ADC_IN4) PA4 -|10      23|- PA13 //connect to JDY40
// (ADC_IN5) PA5 -|11      22|- PA12
```

```
// (ADC_IN6) PA6 -|12      21|- PA11
// (ADC_IN7) PA7 -|13      20|- PA10 (Reserved for RXD)
// (ADC_IN8) PB0 -|14      19|- PA9  (Reserved for TXD)
// (ADC_IN9) PB1 -|15      18|- PA8  (LED+1k)
//           VSS -|16      17|- VDD
//                 ----------

void Configure_Pins (void)
{
	RCC->IOPENR |= BIT0; // peripheral clock enable for port A

	// Make pins PA0 to PA5 outputs (page 200 of RM0451, two bits used to
configure: bit0=1, bit1=0)
    GPIOA->MODER = (GPIOA->MODER & ~(BIT0|BIT1)) | BIT0; // PA0
	GPIOA->OTYPER &= ~BIT0; // Push-pull

    GPIOA->MODER = (GPIOA->MODER & ~(BIT2|BIT3)) | BIT2; // PA1
	GPIOA->OTYPER &= ~BIT1; // Push-pull

    GPIOA->MODER = (GPIOA->MODER & ~(BIT4|BIT5)) | BIT4; // PA2
	GPIOA->OTYPER &= ~BIT2; // Push-pull

    GPIOA->MODER = (GPIOA->MODER & ~(BIT6|BIT7)) | BIT6; // PA3
	GPIOA->OTYPER &= ~BIT3; // Push-pull

    GPIOA->MODER = (GPIOA->MODER & ~(BIT8|BIT9)) | BIT8; // PA4
	GPIOA->OTYPER &= ~BIT4; // Push-pull

    GPIOA->MODER = (GPIOA->MODER & ~(BIT10|BIT11)) | BIT10; // PA5
	GPIOA->OTYPER &= ~BIT5; // Push-pull


   // GPIOA->MODER  = (GPIOA->MODER & ~(BIT17|BIT16) ) | BIT16; // Make pin PA8
output (page 200 of RM0451, two bits used to configure: bit0=1, bit1=0))

	// Configure the pin used for analog input: PB1 (pin 15)
	RCC->IOPENR  |= BIT1;         // peripheral clock enable for port B
	GPIOA->MODER |= (BIT14|BIT15);  // Select analog mode for PB1 (pin 15 of
LQFP32 package)

	GPIOB->MODER |= (BIT0|BIT1);
}

void Button_Init(void)
{
    RCC->IOPENR |= BIT1;  // Enable clock for Port B if not already enabled

    // Configure PB7 (pin30) as input:
    // For PB7, the mode bits are at positions (7 * 2) and (7 * 2 + 1), i.e., bits
14 and 15.
    GPIOB->MODER &= ~(3 << (7 * 2)); // Clear mode bits for PB7 (set as input)
    // Configure PB7 with internal pull-up (set PUPDR bits to '01')
    GPIOB->PUPDR &= ~(3 << (7 * 2));
    GPIOB->PUPDR |= (1 << (7 * 2));
```

```
    // Configure PB6 (pin29) as input:
    // For PB6, the mode bits are at positions (6 * 2) and (6 * 2 + 1), i.e., bits
12 and 13.
    GPIOB->MODER &= ~(3 << (6 * 2)); // Clear mode bits for PB6 (set as input)
    // Configure PB6 with internal pull-up (set PUPDR bits to '01')
    GPIOB->PUPDR &= ~(3 << (6 * 2));
    GPIOB->PUPDR |= (1 << (6 * 2));

       // Configure PB4 (assumed to be pin27) as input with internal pull-up
       // For PB4, mode bits are located at positions (4 * 2) and (4 * 2 + 1)
       GPIOB->MODER &= ~(3 << (4 * 2));  // Clear mode bits for PB4 (set as input)
       GPIOB->PUPDR &= ~(3 << (4 * 2));    // Clear pull-up/pull-down bits for PB4
       GPIOB->PUPDR |= (1 << (4 * 2));     // Set PB4 to pull-up

}



void Hardware_Init(void)
{
       GPIOA->OSPEEDR=0xffffffff; // All pins of port A configured for very high
speed! Page 201 of RM0451

       RCC->IOPENR |= BIT0; // peripheral clock enable for port A

    GPIOA->MODER = (GPIOA->MODER & ~(BIT27|BIT26)) | BIT26; // Make pin PA13 output
(page 200 of RM0451, two bits used to configure: bit0=1, bit1=0))
       GPIOA->ODR |= BIT13; // 'set' pin to 1 is normal operation mode.

}

void Buzzer_Init(void)
{
       // Set up output port bit for blinking LED
       RCC->IOPENR |= 0x00000001; // peripheral clock enable for port A
    GPIOA->MODER  = (GPIOA->MODER & ~(BIT17|BIT16) ) | BIT16;
       // Make pin PA0 output (page 172, two bits used to configure: bit0=1, bit1=0)

       // Set up timer
       RCC->APB2ENR |= BIT2;  // turn on clock for timer21 (UM: page 188)
       //TIM21->ARR = SYSCLK/TICK_FREQ;
       NVIC->ISER[0] |= BIT20; // enable timer 21 interrupts in the NVIC
       TIM21->CR1 |= BIT4;      // Downcounting
       TIM21->CR1 |= BIT0;      // enable counting
       TIM21->DIER |= BIT0;     // enable update event (reload event) interrupt
       __enable_irq();
       TIM21->ARR = 0;
}

void wait_1ms(void)
{
       // For SysTick info check the STM32l0xxx Cortex-M0 programming manual.
       SysTick->LOAD = (F_CPU/1000L) - 1;  // set reload register, counter rolls over
```

```
from zero, hence -1
	SysTick->VAL = 0; // load the SysTick counter
	//SysTick->CTRL = 0x05; // Bit 0: ENABLE, BIT 1: TICKINT, BIT 2:CLKSOURCE
	SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk; //
Enable SysTick IRQ and SysTick Timer */
	while((SysTick->CTRL & BIT16)==0); // Bit 16 is the COUNTFLAG.  True when
counter rolls over from zero.
	SysTick->CTRL = 0x00; // Disable Systick counter
}



void delayms(int len)
{
	while(len--) wait_1ms();
}

void SendATCommand (char * s)
{
	char buff[40];
	printf("Command: %s", s);
	GPIOA->ODR &= ~(BIT13); // 'set' pin to 0 is 'AT' mode.
	waitms(10);
	eputs2(s);
	egets2(buff, sizeof(buff)-1);
	GPIOA->ODR |= BIT13; // 'set' pin to 1 is normal operation mode.
	waitms(10);
	printf("Response: %s", buff);
}
void ReceptionOff (void)
{
	GPIOA->ODR &= ~(BIT13); // 'set' pin to 0 is 'AT' mode.
	waitms(10);
	eputs2("AT+DVID0000\r\n"); // Some unused id, so that we get nothing in RXD1.
	waitms(10);
	GPIOA->ODR |= BIT13; // 'set' pin to 1 is normal operation mode.
	while (ReceivedBytes2()>0) egetc2(); // Clear FIFO
}

void send_jdy (void)
{
	char buff[80];
	eputc2('!'); // Send a message to the slave. First send the 'attention'
character which is '!'
	// Wait a bit so the slave has a chance to get ready
	waitms(50); // This may need adjustment depending on how busy is the slave
	eputs2(buff); // Send the test message
	waitms(50);

	eputc2('@');

	//printf("Slave says: %s\r\n", buff);

}
```

```
void main(void)
{
	float a;	//analog input for x direction (joystick)
	float b;	//analog input for y direction (joystick)
	int adcx;
	int adcy;
	int int_a;	//change a to integer
	int int_b;	//change b to integer
	char buff[800];
	char lcd_b[17];
    int timeout_cnt=0;
    int cont1=0, cont2=100;
    int flag_w = 0;		//flag to check if it is in welcome state
	int flag_m = 0;		//flag to check the page of menu
	int flag_s = 0;		//flag for mode chosing
	int flag_a = 0;	//flag to start auto picking
	int flag_b = 0;		//flag to check if the button of joystick has been
pressed
	int flag_o = 0;
	int cnt = 0;
	char c;
	int buzz;
	int buzz_pre;
	int buzz_sound;
	int buzz_base = 0;
	int coin_count = 0;
	int strength = 0;


    Buzzer_Init();
	Configure_Pins();
	initADC();
	LCD_4BIT();
	Button_Init();
	Hardware_Init();
	initUART2(9600);

    delayms(500); // Give PuTTY time to start
	printf("\x1b[2J\x1b[1;1H"); // Clear screen using ANSI escape sequence.
	printf("STM32L051 ADC Test.  Analog input is PB1 (pin 15).\r\n");


	waitms(1000); // Give putty some time to start.
	printf("\r\nJDY-40 Master test\r\n");



	ReceptionOff();
	//To check configuration
	SendATCommand("AT+VER\r\n");
	SendATCommand("AT+BAUD\r\n");
	SendATCommand("AT+RFID\r\n");
	SendATCommand("AT+DVID\r\n");
	SendATCommand("AT+RFC\r\n");
```

```
        SendATCommand("AT+POWE\r\n");
        SendATCommand("AT+CLSS\r\n");
        // We should select an unique device ID.  The device ID can be a hex
        // number from 0x0000 to 0xFFFF.  In this case is set to 0xABBA
        SendATCommand("AT+DVIDCBCB\r\n");
        cnt=0;
while(1)
{

        while(flag_w == 0)
    {
         LCDprint("Welcome to      ", 1, 1);
         LCDprint("Project 2 demo  ", 2, 1);
         if(!(GPIOB->IDR & BIT7))
         {
             waitms(50);  // Debounce delay
             if(!(GPIOB->IDR & BIT7))
             {
                 flag_w = 1;
             }
         }

    }

    // flag_w = 1, in mode choosing state
    while(flag_w == 1)
    {
             //flag_m = 0 , in fisrt page of mode
         while(flag_m == 0)
         {
             LCDprint("Mode:           ", 1, 1);
             LCDprint("a.Auto picking  ", 2, 1);
             if(!(GPIOB->IDR & BIT7))
             {
                 waitms(50);  // Debounce delay
                 if(!(GPIOB->IDR & BIT7))
                 {
                     flag_m = 1;
                 }
             }
                     if(!(GPIOB->IDR & BIT6))
             {
                 waitms(50);  // Debounce delay
                 if(!(GPIOB->IDR & BIT6))
                 {
                     flag_m = 3;
                                     flag_s = 1;
                                     flag_w = 2;
                                     flag_a = 0;
                 }
             }
         }
             //flag_m = 1 , in second page of mode
         while(flag_m == 1)
```

```
    {
                LCDprint("Mode:           ", 1, 1);
        LCDprint("b.Manual picking", 2, 1);
        if(!(GPIOB->IDR & BIT7))
        {
            waitms(50);  // Debounce delay
            if(!(GPIOB->IDR & BIT7))
            {
                flag_m = 2;
            }
        }
                if(!(GPIOB->IDR & BIT6))
        {
            waitms(50);  // Debounce delay
            if(!(GPIOB->IDR & BIT6))
            {
                flag_m = 3;
                              flag_s = 2;
                              flag_w = 2;
            }
        }
    }
         //flag_m = 2 , in third page of mode
         while(flag_m == 2)
    {
                LCDprint("Mode:           ", 1, 1);
        LCDprint("c.obstacle avoid", 2, 1);
        if(!(GPIOB->IDR & BIT7))
        {
            waitms(50);  // Debounce delay
            if(!(GPIOB->IDR & BIT7))
            {
                              flag_m = 0;
            }
        }
                if(!(GPIOB->IDR & BIT6))
        {
            waitms(50);  // Debounce delay
            if(!(GPIOB->IDR & BIT6))
            {
                flag_m = 3;
                              flag_s = 3;
                              flag_w = 2;
                              flag_o = 0;
            }
        }
    }
}
// Start the whole programe
   //flag_s = 1 for mode 1:aoto picking
   //flag_s = 2 for mode 2:manual picking
   //flag_s = 3 for mode 3:wait to add(bonus)

   //mode 1: auto picking
```

```
	while(flag_s == 1){
		buzz = 0;
		if(!(GPIOB->IDR & BIT4))
	  {
		   waitms(50);  // Debounce delay
		   if(!(GPIOB->IDR & BIT4))
		   {
		       flag_a = 1;
		   }
	  }
		if(!(GPIOB->IDR & BIT7))
	  {
		   waitms(50);  // Debounce delay
		   if(!(GPIOB->IDR & BIT7))
		   {
		       flag_s = 0;
		                     flag_w = 1;
		                     flag_m = 0;
		                     flag_a = 3;
		   }
	  }
		if(flag_a == 0){
			LCDprint("Mode:Auto picking", 1, 1);
		LCDprint("Press to start    ", 2, 1);
		}
		else if(flag_a == 1){

			LCDprint("Start!", 1, 1);
			waitms(200);
			LCDprint("                 ", 1, 1);
		LCDprint("                 ", 2, 1);
			sprintf(buff, "%03d %03d\n", -20, -20);
			eputc2('!'); // Send a message to the slave. First send the 
'attention' character which is '!'
			// Wait a bit so the slave has a chance to get ready
			waitms(50); // This may need adjustment depending on how busy is 
the slave
			eputs2(buff); // Send the test message
			flag_a = 2;
			//}


		}
		else if(flag_a == 2){
			if(coin_count <= 20){
				sprintf(lcd_b, "COIN NUMBER:%d", coin_count);
				LCDprint(lcd_b, 1, 1);
				sprintf(lcd_b, "strength:%d", strength);
				LCDprint(lcd_b, 2, 1);
			}
			else{
				LCDprint("finish picking", 1, 1);
			}
			eputc2('@');
```

```
                    waitms(50);
                    timeout_cnt=0;
                    while(1)
                    {
                        if(ReceivedBytes2()>5) break; // Something has arrived
                        if(++timeout_cnt>30) break; // Wait up to 25ms for the
repply

                        Delay_us(100); // 100us*250=25ms
                    }

                    if(ReceivedBytes2()>5) // Something has arrived from the slave
                    {
                        egets2(buff, sizeof(buff)-1);
                        if(strlen(buff)==6)
                        {
                            printf("Slave says: %s  \r", buff);
                            sscanf(buff, "%d", &buzz);
                            if(buzz_pre < buzz){
                                coin_count++;
                            }
                            else{
                                coin_count = coin_count;
                            }
                            buzz_pre = buzz;
                            strength = buzz;
                            //printf("%d",buzz);
                        }
                        else
                        {
                            while (ReceivedBytes2()) egetc2(); // Clear FIFO
                            printf("*** BAD MESSAGE ***: %s\r", buff);
                        }
                    }
                    else // Timed out waiting for reply
                    {
                        while (ReceivedBytes2()) egetc2(); // Clear FIFO
                        printf("NO RESPONSE\r\n", buff);
                        buzz_pre = 0;
                    }
                    ///buzzer
                    buzz_sound = buzz - 57400;
                    if(buzz_sound >1100){
                        TIM21->ARR = SYSCLK / (1000L);
                    }
                    else if(buzz_sound<= 1100 && buzz_sound>800){
                        TIM21->ARR = SYSCLK / (1500L);
                    }
                    else if(buzz_sound<= 800 && buzz_sound>500){
                        TIM21->ARR = SYSCLK / (2000L);
                    }
                    else if(buzz_sound<= 500 && buzz_sound>300){
                        TIM21->ARR = SYSCLK / (2500L);
                    }
                    else{
```

```
                        TIM21->ARR = 0;
                    }
                    //TIM21->ARR = buzz;

            }


        }

        //mode 2: manual picking
        while(flag_s == 2)
        {
            if(!(GPIOB->IDR & BIT4))
         {
               waitms(50);  // Debounce delay
               if(!(GPIOB->IDR & BIT4))
               {
                   flag_b = 1;
               }
         }
            if(!(GPIOB->IDR & BIT7))
         {
               waitms(100);  // Debounce delay
               if(!(GPIOB->IDR & BIT7))
               {
                   flag_s = 0;
                                flag_w = 1;
                                flag_m = 0;
               }
         }

            adcx=readADC(ADC_CHSELR_CHSEL8);
            a=(adcx*3.3)/0x1000;             //pin9
            a = a*100;

            adcy = readADC(ADC_CHSELR_CHSEL7);
            b = (adcy*3.3)/0x1000;           //pin8
            b = b*100;

            int_a = (int)a;
            int_b = (int)b;


            sprintf(lcd_b, "x= %.3f", a);
            LCDprint(lcd_b, 1, 1);
            sprintf(lcd_b, "y= %.3f", b);
            LCDprint(lcd_b, 2, 1);

            //GPIOA->ODR ^= BIT8; // Complement PA8 (pin 18)
            //delayms(500);
            if(flag_b ==0){
                sprintf(buff, "%03d %03d\n", int_a, int_b); // Construct a test
message
            }
```

```
	else if(flag_b == 1){
		sprintf(buff, "%03d %03d\n", -10, -10); // Construct a test
message, send -10 if the button for arm is pressed

		flag_b = 0;
	}

	eputc2('!'); // Send a message to the slave. First send the 'attention'
character which is '!'
	// Wait a bit so the slave has a chance to get ready
	waitms(50); // This may need adjustment depending on how busy is the
slave
	eputs2(buff); // Send the test message
	waitms(50);

	eputc2('@');

	//printf("Slave says: %s\r\n", buff);
	timeout_cnt=0;
	while(1)
	{
		if(ReceivedBytes2()>5) break; // Something has arrived
		if(++timeout_cnt>30) break; // Wait up to 25ms for the repply
		Delay_us(100); // 100us*250=25ms
	}

	if(ReceivedBytes2()>5) // Something has arrived from the slave
	{
		egets2(buff, sizeof(buff)-1);
		if(strlen(buff)==6)
		{
			printf("Slave says: %s  \r", buff);
		}
		else
		{
			while (ReceivedBytes2()) egetc2(); // Clear FIFO
			printf("*** BAD MESSAGE ***: %s\r", buff);
		}
	}
	else // Timed out waiting for reply
	{
		while (ReceivedBytes2()) egetc2(); // Clear FIFO
		printf("NO RESPONSE\r\n", buff);
	}


	//Something has arrived
	//printf("V=%fV V=%fV\r", a, b);
	TIM21->ARR = 0;
	fflush(stdout);


}
```

```
        while(flag_s == 3){
               if(!(GPIOB->IDR & BIT7))
          {
                  waitms(50);  // Debounce delay
                  if(!(GPIOB->IDR & BIT7))
                  {
                      flag_s = 0;
                                      flag_w = 1;
                                      flag_m = 0;
                  }
          }
               if(flag_o = 0){
                      sprintf(buff, "%03d %03d\n", -30, -30);
                      eputc2('!'); // Send a message to the slave. First send the
'attention' character which is '!'
                      // Wait a bit so the slave has a chance to get ready
                      waitms(50); // This may need adjustment depending on how busy is
the slave
                      eputs2(buff); // Send the test message
                      flag_o = 0;
               }
               LCDprint("Auto: obstacle    ", 1, 1);
          LCDprint("      avoidance   ", 2, 1);
        }

}

}
```