UM–SJTU Joint Institute

Embedded System Design (VE373)

(Design of Microprocessor Based Systems)

Project Report

Hand Gesture Controlled Mantis

Group 8

Name: Yihua Liu ID: 518021910998
Name: Xingyuan Wang ID: 518370910198
Name: Haorong Lu ID: 518370910194

Date: August 5, 2021

Contents

1 Objectives

- Implement a mantis-like robot that can move forward, backward, to the left, and to the right.
- Implement remote control for the robot by hand gestures through Bluetooth.
- Implement the function of obstacle avoidance for the robot.

2 Introduction

2.1 Overview

In this project, we have designed and implemented a robot which can be controlled remotely using gestures. The robot uses 2 feet with 4 servos to move with another 2 wheels for additional supporting, and has the basic functions including standing, moving forward, moving backward, turning clockwise, and turning counterclockwise. The movement of the robot is controlled by hand gestures using an accelerometer-integrated glove, and the communication between the control side and the robot is established via Bluetoorh modules. Two PIC32 boards are used in the project, with one on the robot controlling the movement of the robot, and the other connected to the glove detecting and sending gesture control signals to the robot. Another additional Arduino board is used for connecting the accelerometer with PIC32 via I2C. A standalone battery and a power management IC is used for the power supply of the 4 servos on the robot, and an ultrasonic distance detector is placed on the robot so that is stops moving forward automatically when there are obstacles in front of it.

2.2 Mechanical Structure

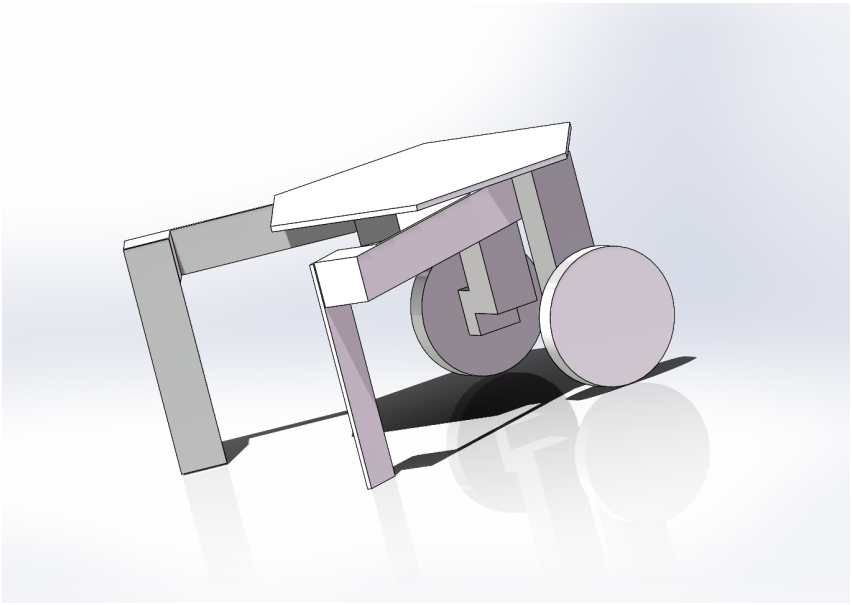


Figure 1. Mechanical Structure of the Robot.

2.3 Photos of the Designed System

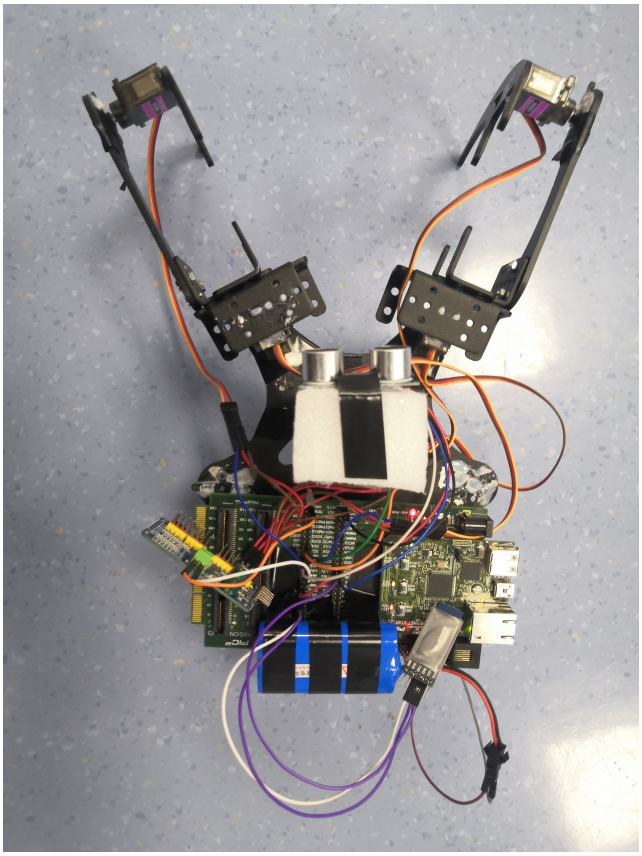


Figure 2. Top View of the Robot.

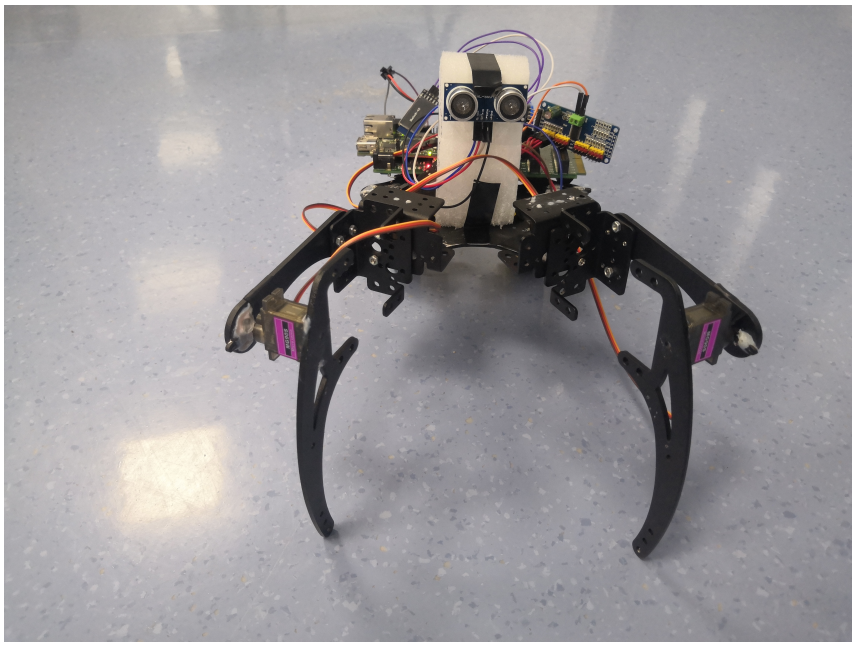


Figure 3. Front View of the Robot.

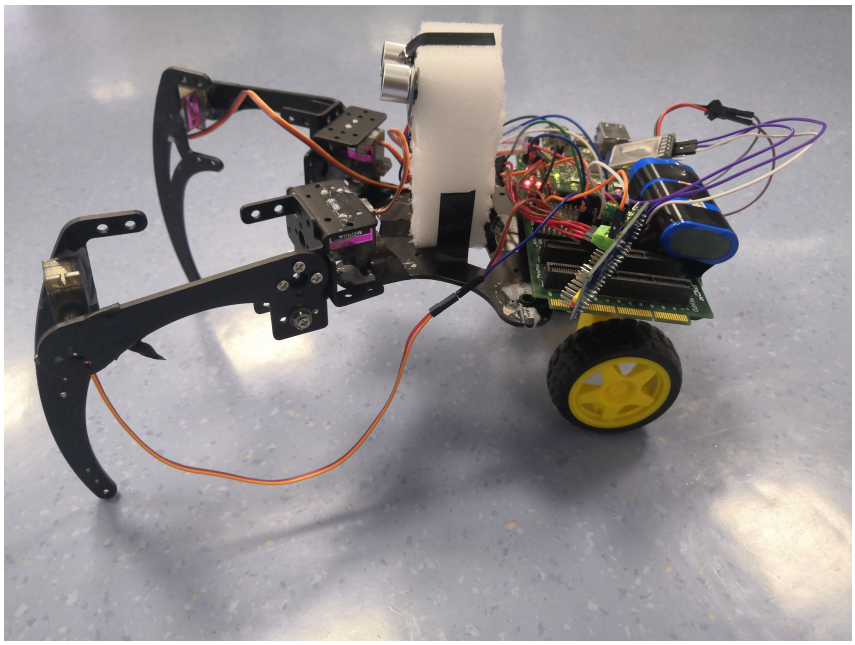


Figure 4. Left View of the Robot.

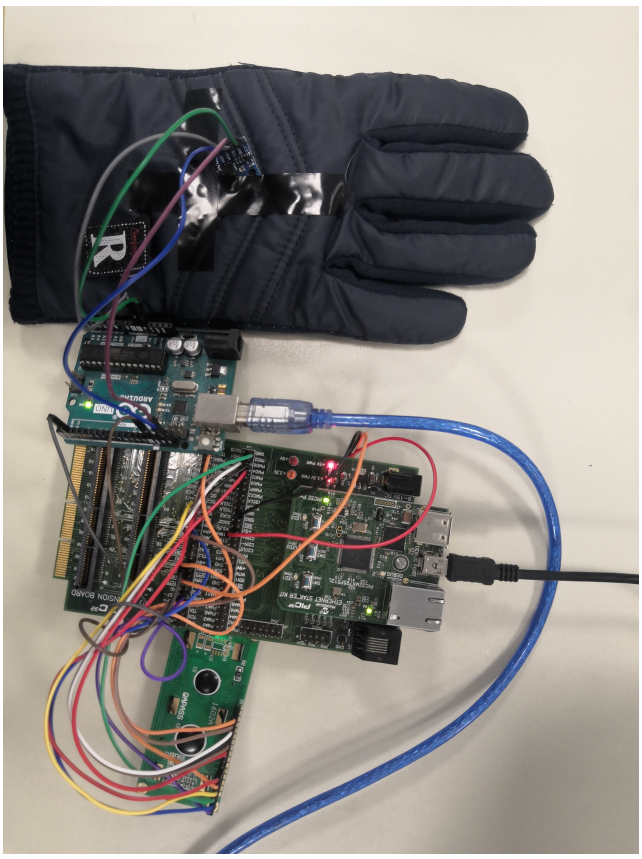


Figure 5. The Circuits of the Remote Control Part.

2.4 Top-level Block Diagram

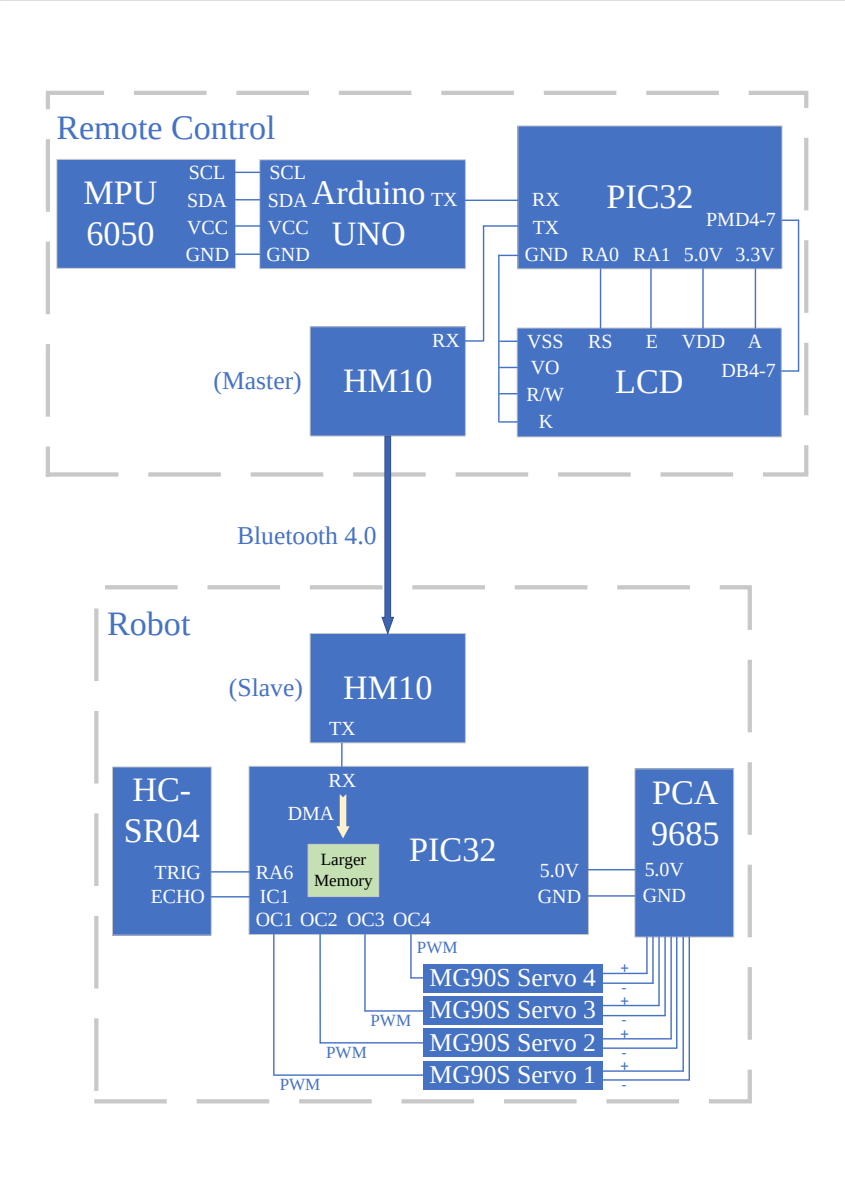


Figure 6. Top-level Block Diagram.

3 Schematics

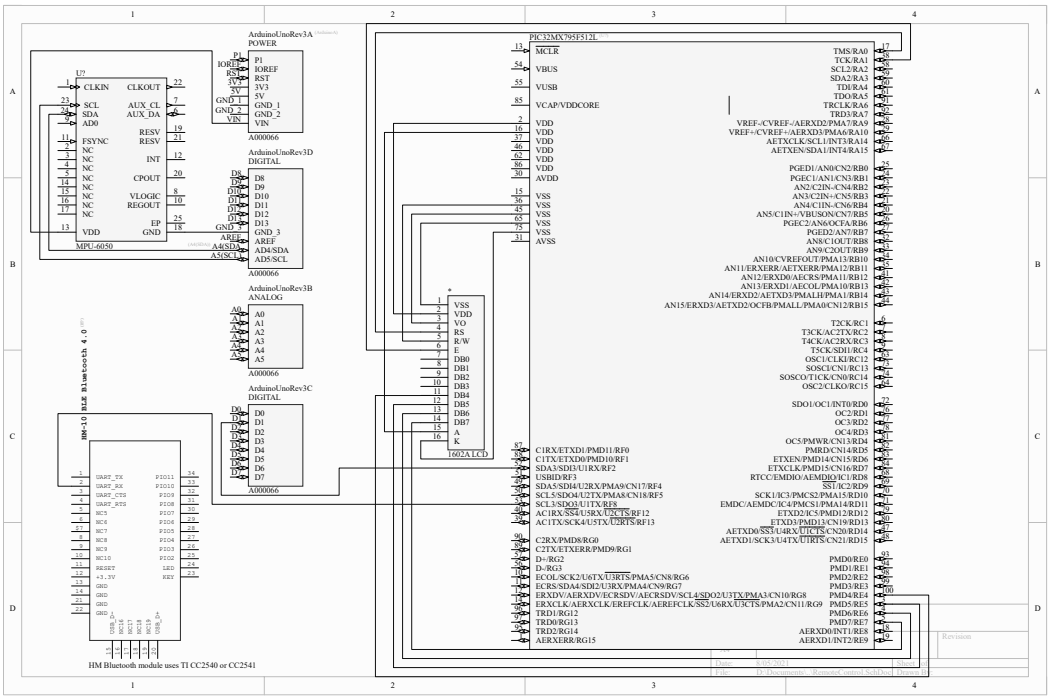


Figure 7. Remote Control Schematic[1].

The original print of the schematic is appended in Appendix.

4 Component Diagram

This section covers the main peripheral components used in our project.

4.1 Motion Tracking Device

In this project, we use MPU-6050, which is an six DoF accelerometer and gyroscope as our motion tracking device for remote control using hands. It measures temperature, acceleration and gravity data in x, y and z axis directions, and communicates with our microprocessor using I2C protocol. The pins used in our program have been labelled in the following diagram.

[1]The library of HM-10 BLE Blootooth 4.0 refers to [1].

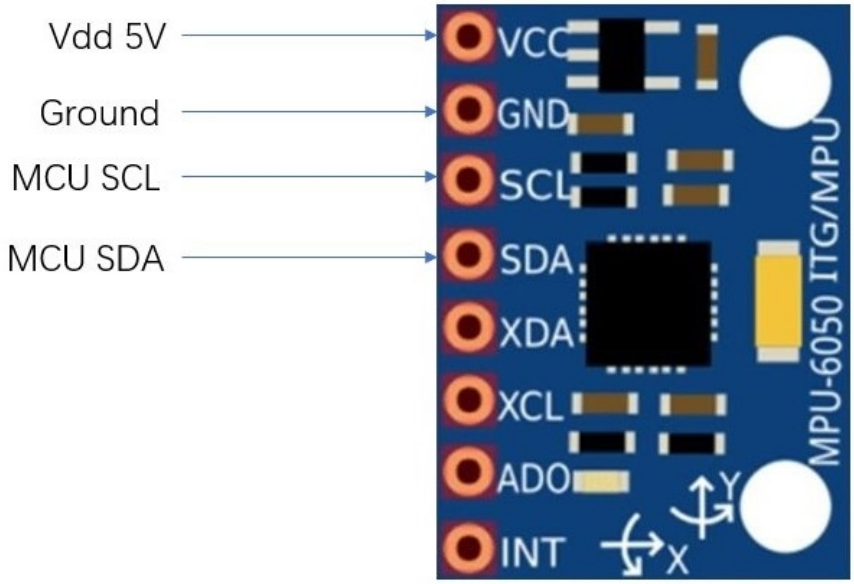


Figure 8. Component diagram of MPU-6050.

However, after some attempts, we find that the I2C module of the PIC32 board cannot work together with the MPU-6050. Our solution is to use an Arduino Uno board as the "intermediary" between MPU-6050 and the PIC32 board. As shown in the functional block diagram in the previous section, the MPU-6050 communicates with the Arduino Uno board through the I2C protocol, and the Arduino Uno board communicates with the PIC32 board through UART protocol. For the MPU-6050, its VCC pin is connected to the $5V$ output of the Arduino Uno board, and the GND pin is connected to the same ground as Arduino Uno. MPU-6050's SCL and SDA pins are connected to the SCL and SDA pins of the Arduino Uno board. In the I2C communication, the MPU-6050 operates in a slave mode while the Arduino Uno operates in the master mode. The other four pins of MPU-6050 are for auxiliary data, clock, address transportation and interrupt, which are not needed in our case.

For the Arduino Uno board, we connect its V_{in} and GND pins to the $5V$ output and GND pins of the PIC32 board, and its UART Tx pin to the U1RX pin of the PIC32 board, because in our situation, we only a need one-way connection from the Arduino Uno board to the PIC32 board.

4.2 Bluetooth Module

We choose the HM-10 Bluetooth module in our project to establish the remote connection between the hand motor detector and the robot. We use the Bluetooth module on the hand motor detector as the transmitter and use the Bluetooth module on the robot as the receiver, with motion instructions being transmitted. Similar to the communication between the Arduino Uno board and the PIC32 board, we only need to implement the one-way communication between the hand motion detector and the robot.

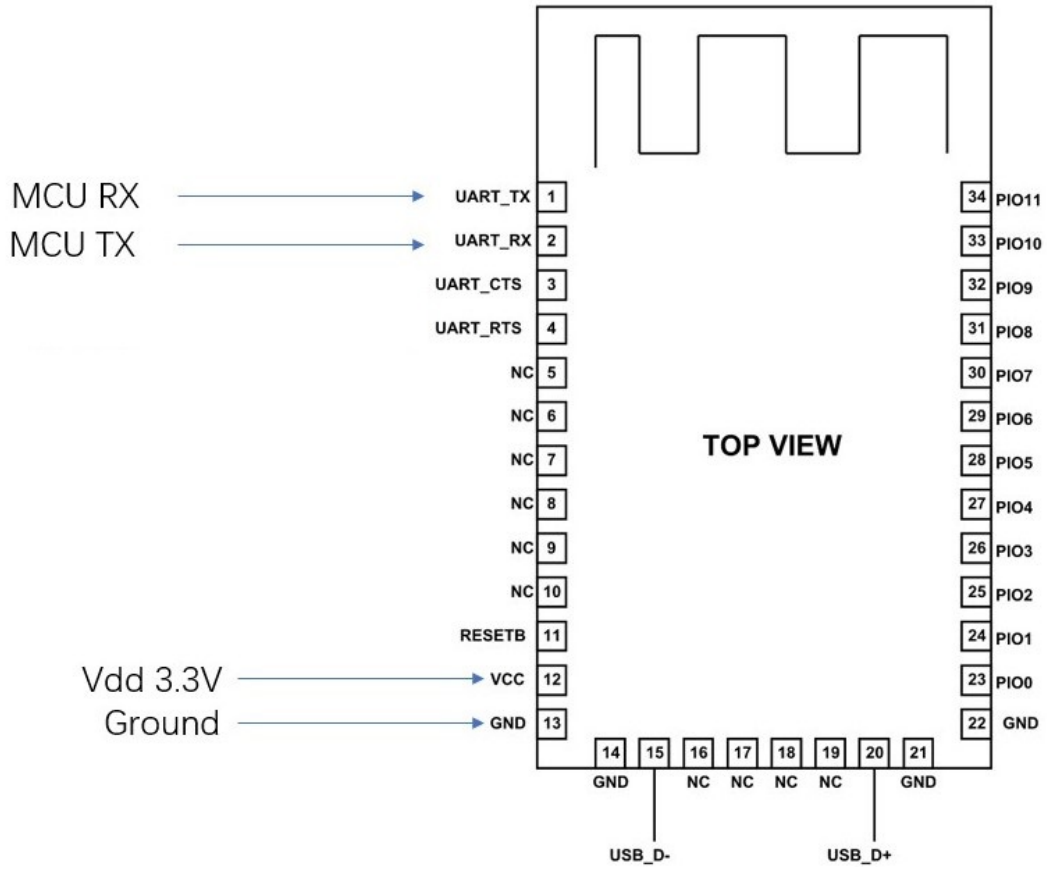


Figure 9. Component diagram of HM-10.

The configuration of the Bluetooth modules is done manually. We connect each Bluetooth module to a computer by using a USB to TTL converter and an application which supports serial communication. We send a 'AT+MODE0' message to each Bluetooth module to put them into the data transmission mode. Then we send 'AT+ROLE1' to the Bluetooth module on the hand motion detector to put it into master mode, and send 'AT+ROLE0' to the Bluetooth module on the robot to put it into slave mode. The HM-10 Bluetooth module supports auto-connection, namely a Bluetooth in master mode can automatically detect nearby Bluetooth module that is in slave mode, and connect to it. Therefore, there is no need to setup the connection manually every time, we just need to turn on the power, and the two Bluetooth modules will connect each other automatically.

As only one-way data transmission is used in our project, only 3 pins labelled in the diagram is needed. The Bluetooth module is connected to our PIC32 via UART with a required baud rate being 9600. On the hand motion detector, we connect two pins for power supply, and then connect the UART Rx pin of the Bluetooth module to the U1TX pin of the PIC32 board. On the robot, we also connect two pins for power supply, and then connect the UART Tx pin of the Bluetooth module to the U1RX pin of the PIC32 board. Also, we use DMA to transmit received data to a larger memory space as soon as the receiver has data available.

4.3 Servo

In this project, we use 4 servos to control the motion of the 2 front legs of our spider, since each leg needs to move upward, downward, forward and backward. We choose MG90S as our servo, which has a rotating angle between 0 and 180 degrees controlled by PWM

signals. The required frequency of the PWM signal is 50Hz. The following diagram shows the pinout of the servo.

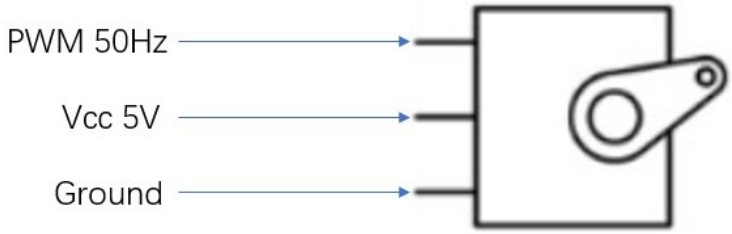


Figure 10. Component diagram of MG90S.

As can be seen from the diagram, we need one PWM signal per servo which makes a total of 4 PWM signals. The PIC32 board has five output compare modules OC1-OC5, and that's enough for us to use OC to generate the PWM signals for 4 servos. The relationship between the duty cycle of PWM signal and the degree of servo is shown in the table below.

Duty Cycle	MG90S Servo Degree
2.5%	0°
5.0%	45°
7.5%	90°
10.0%	135°
12.5%	180°

Table 1: Duty Cycle v.s. MG90S Servo Degree

Besides the PWM signal generating, another problem is how to power these servos. The Arduino motor shield we used in the lab can only support the power supply for 2 servos. Therefore, we purchase a PCA9685 board that can support the power supply for at most 16 servos. The following diagram shows the appearance of the PCA9685 board.

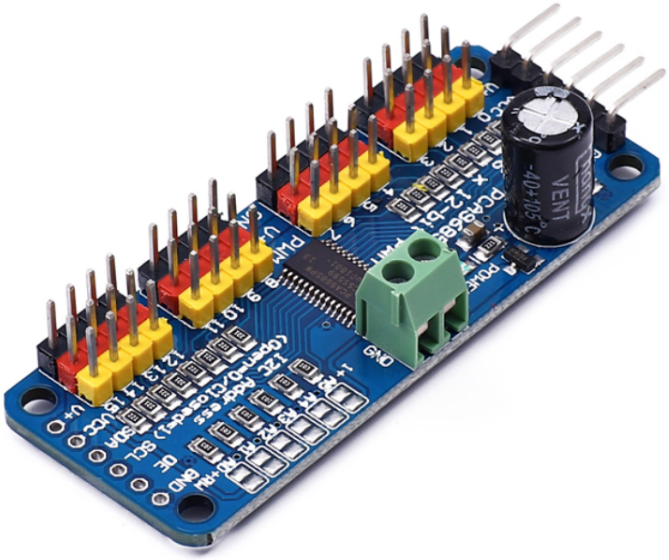


Figure 11. Appearance of the PCA9685 board.

On our robot, we connect the $5V$ output and GND of the PIC32 board to the V_{in} and GND (the green part in the figure above) of the PCA9685, and connect each servo to one pair of the V+ and GND pins (the red and black part) respectively.

4.4 Distance Detector

We plan to add an ultrasonic distance sensor HC-SR04 on our spider facing forward which detects the distance between the spider and potential obstruction in its front. The diagram of HC-SR04 is shown below.

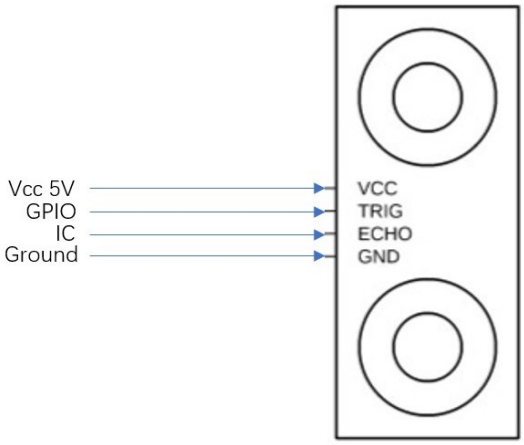


Figure 12. Component diagram of HC-SR04.

The GPIO pin of PIC32 connected to TRIG pin in HC-SR04 sends a high value exceeding $10\mu s$ to trigger the distance measurement. If any signal comes back to the module, the distance will be proportional to time of high value on ECHO pin. In this case, we use an input capture module to measure that time of high value on ECHO pin. When the distance is below a certain value, the spider is expected to refuse moving forward.

5 Tests and Results

Our test plan generally has three phases, we first do a single unit test on each component, and then test the hand motion detector and the robot respectively. Finally we connect this two parts together, and test the overall operating status.

5.1 MPU-6050

In the program on the Arduino Uno board, we print the data received from MPU-6050, and use Arduino's serial monitor to show the results. It appears that the data we get for each hand motion (stay and flip left/right/forward/backword) confirms to the datasheet description and can be separable.

5.2 HM-10 Bluetooth Module

We use USB to TTL converter to connect each HM-10 to one computer, and use serial communication application to control and test the HM-10 modules. We first send 'AT+ROLE0' to one HM-10 and 'AT+ROLE1' to another HM-10, and they both send back 'OK:SET+0/1', which indicates that they have correctly accepted the AT command. Then the red lights on both HM-10 change from flashing to steady, which shows that the auto-connection function of two modules works well. Finally we send a message from one HM-10

to another HM-10 module through the serial communication application, and another HM-10 correctly receives the same message. We can conclude that our Bluetooth module works well.

5.3 Servo

We directly connect one servo to the PIC32 board and write a simple program that generates a PWM signal with different duty cycles to test it. It appears that the servo's behavior conforms to the relationship between the duty cycle and servo's degree in the previous section, and it's strong enough to rotate the whole leg.

5.4 Distance Detector

The test plan of the distance detector is similar to the test plan of MPU-6050. We connect it to the Arduino Uno board and use the serial monitor to show its result. It appears that the HC-SR04 ultrasonic sensor can reflect properly the changes in the distance between it and the nearest obstacle.

5.5 Hand Motion Detector

After testing the MPU-6050 and HM-10, we connect them together to test te whole hand motion detector. We connect an LCD screen to the PIC32 board so that we can check the data it receives. We test five different hand motions, and the result on the LCD and arduino's serial monitor shows that the Arduino Uno board correctly process the raw data from the MPU-6050 and generate the direction, and the PIC32 board can properly receive the direction.

5.6 Robot

After we finish the configuration of each servo, assembling the robot, and determining the moving algorithm, we start to test the entire moving function of the robot. The test plan is quite simple, we manually set the direction we want the robot to move, and check its behavior. As shown in our video, its moving behavior conforms to our expectation.

5.7 Overall Test

Since the hand motion detector and robot work well, we only have to connect them together through Bluetooth, and check the overall behavior. As shown in our video, we can see that the robot moves properly according to our hand motion, and the delay between these two parts is short. In addition, when the distance detector find there is an obstacle close to it, the robot will stop moving forward to protect its legs. Generally, we are satisfied with the performance of our robot.

6 Final Material List

The final materials we used for the project is shown in Table 2. Compared with our proposal, we significantly reduce the numbers of MG90S servos since we update the mechanical structure. The detailed usages of these materials are introduced in the previous section.

Part	Part Number	Price (rmb)
PIC32 Board (x2)	PIC32MX795F512L	N/A
Accelerometer	MPU6050 6DOF	14
Bluetooth Tx/Rx	HM-10	42
PWM Servo Driver	PCA9685	22
Micro Servo (x4)	MG90S	42
Lipo Battery (x2)	6.4V 1800mAh	70
Ultrasonic Sensor	HCSR04	6
Bracket of Robot	Heaxpod Spyder	200
USB to TTL Converter	CH340G	30
Small Wheel (x2)	65mm diameter	5
Dupont Lines	N/A	0
Total Price		431

Table 2: Final Material List for our Project

7 Final Project Timeline

	Task Name	Duration	Start	End	Completion
1	**Final Project**	**23.12 days**	**2021/7/12**	**2021/8/2**	**100%**
2	Overall Design	7 days	2021/7/12	2021/7/18	100%
3	Purchase Materials	9 days	2021/7/14	2021/7/22	100%
4	**Moving Robot**	**9.5 days**	**2021/7/21**	**2021/7/29**	**100%**
5	Program PIC32 to Control Servos	3 days	2021/7/21	2021/7/23	100%
6	Assemble the Robot	6 days	2021/7/23	2021/7/28	100%
7	Implement Moving Algorithm	2 days	2021/7/28	2021/7/29	100%
8	**Hand Motions Detector**	**4 days**	**2021/7/21**	**2021/7/24**	**100%**
9	Developing Accelerometer Driver	3 days	2021/7/21	2021/7/23	100%
10	Implement Hand Motion Algorithm	2 days	2021/7/23	2021/7/24	100%
11	**Assembly Two Parts**	**4 days**	**2021/7/27**	**2021/7/30**	**100%**
12	Implement Bluetooth Communication	3 days	2021/7/27	2021/7/29	100%
13	Use Hand Motions to Control Robot	2 days	2021/7/29	2021/7/30	100%
14	**Testing and Demonstration**	**4.25 days**	**2021/7/30**	**2021/8/2**	**100%**
15	Testing Prototype and Debug	3 days	2021/7/30	2021/8/1	100%
16	Demonstration	2 days	2021/8/1	2021/8/2	100%

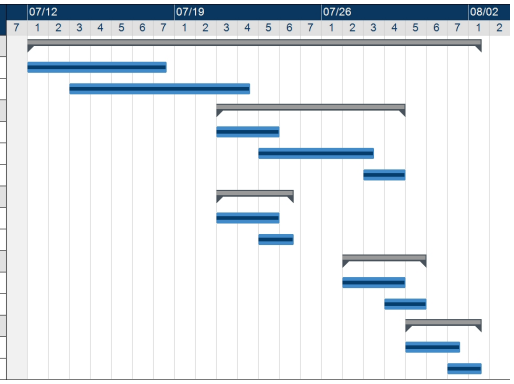


Figure 13. Gantt Chart for our Project.

The final timeline of our project is shown in the figure above. Compared with our proposed timeline, we spent a lot more time on assembling the robot since it's mechanical structure is very complicated, and we have to waste plenty of time on waiting the AB glue to be completely dry. Therefore we update the mechanical structure of the robot by replacing some legs with two wheels, which makes our robot more strong and robust.

Though there were some unexpected difficulties and accidents, we still finished the whole project on August 1st, which gave us some time to prepare for the demonstration (and other courses' final exams).

8 Completion of Project Requirements

- Timer: We use timer to generate delay in the servo driver, and also help generate PWM signal
- Interrupt: We use interrupt from TMR1 to generate delay, and on the PIC32 board for the hand motion detector, we use the DMA interrupt since DMA is used to transfer data sent by the Arduino Uno shield through UART. (The PIC32 board on the robot use polling to deal with the control signal received by the Bluetooth module, since the motion of the robot must be complete to maintain its stability.)

- PWM: We use PWM signal generated by OC1-OC4 to control 4 servos.
- ADC: Not used in our project.
- Serial communication module: We use I2C to support the communication between Arduino Uno and the MPU-6050, and use UART to support the communication between Arduino Uno and PIC32 board, and the communication between PIC32 board and the Bluetooth module.
- DMA: We use DMA on the robot to transmit the received control signals to a larger array, and then we can use some algorithm to find the majority of the control signal sequence.

References

[1] Danilo Díaz Tarascó. *Personal-Altium-Library*. Medellín, Apr. 29, 2019. URL: `https://github.com/ddtdanilo/Personal-Altium-Library`.

Appendix

Code for Remote Control

Arduino Uno

```
/*
    MPU6050 Triple Axis Gyroscope & Accelerometer. Simple Accelerometer
↪   Example.
    Read more: http://www.jarzebski.pl/arduino/czujniki-i-sensory/3-osio
↪   wy-zyroskop-i-akcelerometr-mpu6050.html
    GIT: https://github.com/jarzebski/Arduino-MPU6050
    Web: http://www.jarzebski.pl
    (c) 2014 by Korneliusz Jarzebski
*/

#include <Wire.h>
#include "MPU6050.h"
MPU6050 mpu;
char direction;

void setup()
{
  Serial.begin(9600);
  // Serial.println("Initialize MPU6050");

  while (!mpu.begin(MPU6050_SCALE_2000DPS, MPU6050_RANGE_2G))
  {
    // Serial.println("Could not find a valid MPU6050 sensor, check
    ↪  wiring!");
    delay(500);
  }
  // checkSettings();
}

void checkSettings()
```

```
{
  Serial.println();

  Serial.print(" * Sleep Mode:            ");
  Serial.println(mpu.getSleepEnabled() ? "Enabled" : "Disabled");

  Serial.print(" * Clock Source:          ");
  switch (mpu.getClockSource())
  {
    case MPU6050_CLOCK_KEEP_RESET:     Serial.println("Stops the clock
    ↪  and keeps the timing generator in reset"); break;
    case MPU6050_CLOCK_EXTERNAL_19MHZ: Serial.println("PLL with external
    ↪  19.2MHz reference"); break;
    case MPU6050_CLOCK_EXTERNAL_32KHZ: Serial.println("PLL with external
    ↪  32.768kHz reference"); break;
    case MPU6050_CLOCK_PLL_ZGYRO:      Serial.println("PLL with Z axis
    ↪  gyroscope reference"); break;
    case MPU6050_CLOCK_PLL_YGYRO:      Serial.println("PLL with Y axis
    ↪  gyroscope reference"); break;
    case MPU6050_CLOCK_PLL_XGYRO:      Serial.println("PLL with X axis
    ↪  gyroscope reference"); break;
    case MPU6050_CLOCK_INTERNAL_8MHZ:  Serial.println("Internal 8MHz
    ↪  oscillator"); break;
  }

  Serial.print(" * Accelerometer:         ");
  switch (mpu.getRange())
  {
    case MPU6050_RANGE_16G:            Serial.println("+/- 16 g"); break;
    case MPU6050_RANGE_8G:             Serial.println("+/- 8 g"); break;
    case MPU6050_RANGE_4G:             Serial.println("+/- 4 g"); break;
    case MPU6050_RANGE_2G:             Serial.println("+/- 2 g"); break;
  }

  Serial.print(" * Accelerometer offsets: ");
  Serial.print(mpu.getAccelOffsetX());
  Serial.print(" / ");
  Serial.print(mpu.getAccelOffsetY());
  Serial.print(" / ");
  Serial.println(mpu.getAccelOffsetZ());

  Serial.println();
}

void loop()
{
  Vector rawAccel = mpu.readRawAccel();
  if (rawAccel.ZAxis > 10000)
    direction = 'S';
  else if (rawAccel.YAxis > 10000 && rawAccel.XAxis > -10000 &&
  ↪  rawAccel.XAxis < 10000)
    direction = 'B';
  else if (rawAccel.YAxis < -10000 && rawAccel.XAxis > -10000 &&
  ↪  rawAccel.XAxis < 10000)
```

```
    direction = 'F';
  else if (rawAccel.XAxis < -10000 && rawAccel.YAxis > -10000 &&
  ↪  rawAccel.YAxis < 10000)
    direction = 'C';
  else if (rawAccel.XAxis > 10000 && rawAccel.YAxis > -10000 &&
  ↪  rawAccel.YAxis < 10000)
    direction = 'U';
  else direction = 'S';
  Serial.write(direction);
  delay(100);
}
```

```
/*
  MPU6050.h - Header file for the MPU6050 Triple Axis Gyroscope &
↪  Accelerometer Arduino Library.
  Version: 1.0.3
  (c) 2014-2015 Korneliusz Jarzebski
  www.jarzebski.pl
  This program is free software: you can redistribute it and/or modify
  it under the terms of the version 3 GNU General Public License as
  published by the Free Software Foundation.
  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef MPU6050_h
#define MPU6050_h

#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#define MPU6050_ADDRESS             (0x68) // 0x69 when AD0 pin to Vcc

#define MPU6050_REG_ACCEL_XOFFS_H     (0x06)
#define MPU6050_REG_ACCEL_XOFFS_L     (0x07)
#define MPU6050_REG_ACCEL_YOFFS_H     (0x08)
#define MPU6050_REG_ACCEL_YOFFS_L     (0x09)
#define MPU6050_REG_ACCEL_ZOFFS_H     (0x0A)
#define MPU6050_REG_ACCEL_ZOFFS_L     (0x0B)
#define MPU6050_REG_GYRO_XOFFS_H      (0x13)
#define MPU6050_REG_GYRO_XOFFS_L      (0x14)
#define MPU6050_REG_GYRO_YOFFS_H      (0x15)
#define MPU6050_REG_GYRO_YOFFS_L      (0x16)
#define MPU6050_REG_GYRO_ZOFFS_H      (0x17)
```

```
#define MPU6050_REG_GYRO_ZOFFS_L      (0x18)
#define MPU6050_REG_CONFIG            (0x1A)
#define MPU6050_REG_GYRO_CONFIG       (0x1B) // Gyroscope Configuration
#define MPU6050_REG_ACCEL_CONFIG      (0x1C) // Accelerometer
↪  Configuration
#define MPU6050_REG_FF_THRESHOLD      (0x1D)
#define MPU6050_REG_FF_DURATION       (0x1E)
#define MPU6050_REG_MOT_THRESHOLD     (0x1F)
#define MPU6050_REG_MOT_DURATION      (0x20)
#define MPU6050_REG_ZMOT_THRESHOLD    (0x21)
#define MPU6050_REG_ZMOT_DURATION     (0x22)
#define MPU6050_REG_INT_PIN_CFG       (0x37) // INT Pin. Bypass Enable
↪  Configuration
#define MPU6050_REG_INT_ENABLE        (0x38) // INT Enable
#define MPU6050_REG_INT_STATUS        (0x3A)
#define MPU6050_REG_ACCEL_XOUT_H      (0x3B)
#define MPU6050_REG_ACCEL_XOUT_L      (0x3C)
#define MPU6050_REG_ACCEL_YOUT_H      (0x3D)
#define MPU6050_REG_ACCEL_YOUT_L      (0x3E)
#define MPU6050_REG_ACCEL_ZOUT_H      (0x3F)
#define MPU6050_REG_ACCEL_ZOUT_L      (0x40)
#define MPU6050_REG_TEMP_OUT_H        (0x41)
#define MPU6050_REG_TEMP_OUT_L        (0x42)
#define MPU6050_REG_GYRO_XOUT_H       (0x43)
#define MPU6050_REG_GYRO_XOUT_L       (0x44)
#define MPU6050_REG_GYRO_YOUT_H       (0x45)
#define MPU6050_REG_GYRO_YOUT_L       (0x46)
#define MPU6050_REG_GYRO_ZOUT_H       (0x47)
#define MPU6050_REG_GYRO_ZOUT_L       (0x48)
#define MPU6050_REG_MOT_DETECT_STATUS (0x61)
#define MPU6050_REG_MOT_DETECT_CTRL   (0x69)
#define MPU6050_REG_USER_CTRL         (0x6A) // User Control
#define MPU6050_REG_PWR_MGMT_1        (0x6B) // Power Management 1
#define MPU6050_REG_WHO_AM_I          (0x75) // Who Am I

#ifndef VECTOR_STRUCT_H
#define VECTOR_STRUCT_H
struct Vector
{
  float XAxis;
  float YAxis;
  float ZAxis;
};
#endif

struct Activites
{
  bool isOverflow;
  bool isFreeFall;
  bool isInactivity;
  bool isActivity;
  bool isPosActivityOnX;
  bool isPosActivityOnY;
  bool isPosActivityOnZ;
```

```
  bool isNegActivityOnX;
  bool isNegActivityOnY;
  bool isNegActivityOnZ;
  bool isDataReady;
};

typedef enum
{
  MPU6050_CLOCK_KEEP_RESET      = 0b111,
  MPU6050_CLOCK_EXTERNAL_19MHZ  = 0b101,
  MPU6050_CLOCK_EXTERNAL_32KHZ  = 0b100,
  MPU6050_CLOCK_PLL_ZGYRO       = 0b011,
  MPU6050_CLOCK_PLL_YGYRO       = 0b010,
  MPU6050_CLOCK_PLL_XGYRO       = 0b001,
  MPU6050_CLOCK_INTERNAL_8MHZ   = 0b000
} mpu6050_clockSource_t;

typedef enum
{
  MPU6050_SCALE_2000DPS         = 0b11,
  MPU6050_SCALE_1000DPS         = 0b10,
  MPU6050_SCALE_500DPS          = 0b01,
  MPU6050_SCALE_250DPS          = 0b00
} mpu6050_dps_t;

typedef enum
{
  MPU6050_RANGE_16G             = 0b11,
  MPU6050_RANGE_8G              = 0b10,
  MPU6050_RANGE_4G              = 0b01,
  MPU6050_RANGE_2G              = 0b00,
} mpu6050_range_t;

typedef enum
{
  MPU6050_DELAY_3MS             = 0b11,
  MPU6050_DELAY_2MS             = 0b10,
  MPU6050_DELAY_1MS             = 0b01,
  MPU6050_NO_DELAY              = 0b00,
} mpu6050_onDelay_t;

typedef enum
{
  MPU6050_DHPF_HOLD             = 0b111,
  MPU6050_DHPF_0_63HZ           = 0b100,
  MPU6050_DHPF_1_25HZ           = 0b011,
  MPU6050_DHPF_2_5HZ            = 0b010,
  MPU6050_DHPF_5HZ              = 0b001,
  MPU6050_DHPF_RESET            = 0b000,
} mpu6050_dhpf_t;

typedef enum
{
  MPU6050_DLPF_6                = 0b110,
```

```
  MPU6050_DLPF_5                = 0b101,
  MPU6050_DLPF_4                = 0b100,
  MPU6050_DLPF_3                = 0b011,
  MPU6050_DLPF_2                = 0b010,
  MPU6050_DLPF_1                = 0b001,
  MPU6050_DLPF_0                = 0b000,
} mpu6050_dlpf_t;

class MPU6050
{
  public:

    bool begin(mpu6050_dps_t scale = MPU6050_SCALE_2000DPS,
    ↪  mpu6050_range_t range = MPU6050_RANGE_2G, int mpua =
    ↪  MPU6050_ADDRESS);

    void setClockSource(mpu6050_clockSource_t source);
    void setScale(mpu6050_dps_t scale);
    void setRange(mpu6050_range_t range);
    mpu6050_clockSource_t getClockSource(void);
    mpu6050_dps_t getScale(void);
    mpu6050_range_t getRange(void);
    void setDHPFMode(mpu6050_dhpf_t dhpf);
    void setDLPFMode(mpu6050_dlpf_t dlpf);
    mpu6050_onDelay_t getAccelPowerOnDelay();
    void setAccelPowerOnDelay(mpu6050_onDelay_t delay);

    uint8_t getIntStatus(void);

    bool getIntZeroMotionEnabled(void);
    void setIntZeroMotionEnabled(bool state);
    bool getIntMotionEnabled(void);
    void setIntMotionEnabled(bool state);
    bool getIntFreeFallEnabled(void);
    void setIntFreeFallEnabled(bool state);

    uint8_t getMotionDetectionThreshold(void);
    void setMotionDetectionThreshold(uint8_t threshold);
    uint8_t getMotionDetectionDuration(void);
    void setMotionDetectionDuration(uint8_t duration);

    uint8_t getZeroMotionDetectionThreshold(void);
    void setZeroMotionDetectionThreshold(uint8_t threshold);
    uint8_t getZeroMotionDetectionDuration(void);
    void setZeroMotionDetectionDuration(uint8_t duration);

    uint8_t getFreeFallDetectionThreshold(void);
    void setFreeFallDetectionThreshold(uint8_t threshold);
    uint8_t getFreeFallDetectionDuration(void);
    void setFreeFallDetectionDuration(uint8_t duration);

    bool getSleepEnabled(void);
    void setSleepEnabled(bool state);
    bool getI2CMasterModeEnabled(void);
```

```
    void setI2CMasterModeEnabled(bool state);
    bool getI2CBypassEnabled(void);
    void setI2CBypassEnabled(bool state);

    float readTemperature(void);
    Activites readActivites(void);

    int16_t getGyroOffsetX(void);
    void setGyroOffsetX(int16_t offset);
    int16_t getGyroOffsetY(void);
    void setGyroOffsetY(int16_t offset);
    int16_t getGyroOffsetZ(void);
    void setGyroOffsetZ(int16_t offset);

    int16_t getAccelOffsetX(void);
    void setAccelOffsetX(int16_t offset);
    int16_t getAccelOffsetY(void);
    void setAccelOffsetY(int16_t offset);
    int16_t getAccelOffsetZ(void);
    void setAccelOffsetZ(int16_t offset);

    void calibrateGyro(uint8_t samples = 50);
    void setThreshold(uint8_t multiple = 1);
    uint8_t getThreshold(void);

    Vector readRawGyro(void);
    Vector readNormalizeGyro(void);

    Vector readRawAccel(void);
    Vector readNormalizeAccel(void);
    Vector readScaledAccel(void);

  private:
    Vector ra, rg; // Raw vectors
    Vector na, ng; // Normalized vectors
    Vector tg, dg; // Threshold and Delta for Gyro
    Vector th;     // Threshold
    Activites a;   // Activities

    float dpsPerDigit, rangePerDigit;
    float actualThreshold;
    bool useCalibrate;
    int mpuAddress;

    uint8_t fastRegister8(uint8_t reg);

    uint8_t readRegister8(uint8_t reg);
    void writeRegister8(uint8_t reg, uint8_t value);

    int16_t readRegister16(uint8_t reg);
    void writeRegister16(uint8_t reg, int16_t value);

    bool readRegisterBit(uint8_t reg, uint8_t pos);
    void writeRegisterBit(uint8_t reg, uint8_t pos, bool state);
```

```
};

#endif
```

```
/*
  MPU6050.cpp - Class file for the MPU6050 Triple Axis Gyroscope &
↪  Accelerometer Arduino Library.
  Version: 1.0.3
  (c) 2014-2015 Korneliusz Jarzebski
  www.jarzebski.pl
  This program is free software: you can redistribute it and/or modify
  it under the terms of the version 3 GNU General Public License as
  published by the Free Software Foundation.
  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.
  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/

#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif

#include <Wire.h>
#include <math.h>

#include "MPU6050.h"

bool MPU6050::begin(mpu6050_dps_t scale, mpu6050_range_t range, int mpua)
{
  // Set Address
  mpuAddress = mpua;

  Wire.begin();

  // Reset calibrate values
  dg.XAxis = 0;
  dg.YAxis = 0;
  dg.ZAxis = 0;
  useCalibrate = false;

  // Reset threshold values
  tg.XAxis = 0;
  tg.YAxis = 0;
  tg.ZAxis = 0;
  actualThreshold = 0;
```

```
  // Check MPU6050 Who Am I Register
  if (fastRegister8(MPU6050_REG_WHO_AM_I) != 0x68)
  {
    return false;
  }

  // Set Clock Source
  setClockSource(MPU6050_CLOCK_PLL_XGYRO);

  // Set Scale & Range
  setScale(scale);
  setRange(range);

  // Disable Sleep Mode
  setSleepEnabled(false);

  return true;
}

void MPU6050::setScale(mpu6050_dps_t scale)
{
  uint8_t value;

  switch (scale)
  {
    case MPU6050_SCALE_250DPS:
      dpsPerDigit = .007633f;
      break;
    case MPU6050_SCALE_500DPS:
      dpsPerDigit = .015267f;
      break;
    case MPU6050_SCALE_1000DPS:
      dpsPerDigit = .030487f;
      break;
    case MPU6050_SCALE_2000DPS:
      dpsPerDigit = .060975f;
      break;
    default:
      break;
  }

  value = readRegister8(MPU6050_REG_GYRO_CONFIG);
  value &= 0b11100111;
  value |= (scale << 3);
  writeRegister8(MPU6050_REG_GYRO_CONFIG, value);
}

mpu6050_dps_t MPU6050::getScale(void)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_GYRO_CONFIG);
  value &= 0b00011000;
  value >>= 3;
```

```
  return (mpu6050_dps_t)value;
}

void MPU6050::setRange(mpu6050_range_t range)
{
  uint8_t value;

  switch (range)
  {
    case MPU6050_RANGE_2G:
      rangePerDigit = .000061f;
      break;
    case MPU6050_RANGE_4G:
      rangePerDigit = .000122f;
      break;
    case MPU6050_RANGE_8G:
      rangePerDigit = .000244f;
      break;
    case MPU6050_RANGE_16G:
      rangePerDigit = .0004882f;
      break;
    default:
      break;
  }

  value = readRegister8(MPU6050_REG_ACCEL_CONFIG);
  value &= 0b11100111;
  value |= (range << 3);
  writeRegister8(MPU6050_REG_ACCEL_CONFIG, value);
}

mpu6050_range_t MPU6050::getRange(void)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_ACCEL_CONFIG);
  value &= 0b00011000;
  value >>= 3;
  return (mpu6050_range_t)value;
}

void MPU6050::setDHPFMode(mpu6050_dhpf_t dhpf)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_ACCEL_CONFIG);
  value &= 0b11111000;
  value |= dhpf;
  writeRegister8(MPU6050_REG_ACCEL_CONFIG, value);
}

void MPU6050::setDLPFMode(mpu6050_dlpf_t dlpf)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_CONFIG);
  value &= 0b11111000;
```

```
  value |= dlpf;
  writeRegister8(MPU6050_REG_CONFIG, value);
}

void MPU6050::setClockSource(mpu6050_clockSource_t source)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_PWR_MGMT_1);
  value &= 0b11111000;
  value |= source;
  writeRegister8(MPU6050_REG_PWR_MGMT_1, value);
}

mpu6050_clockSource_t MPU6050::getClockSource(void)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_PWR_MGMT_1);
  value &= 0b00000111;
  return (mpu6050_clockSource_t)value;
}

bool MPU6050::getSleepEnabled(void)
{
  return readRegisterBit(MPU6050_REG_PWR_MGMT_1, 6);
}

void MPU6050::setSleepEnabled(bool state)
{
  writeRegisterBit(MPU6050_REG_PWR_MGMT_1, 6, state);
}

bool MPU6050::getIntZeroMotionEnabled(void)
{
  return readRegisterBit(MPU6050_REG_INT_ENABLE, 5);
}

void MPU6050::setIntZeroMotionEnabled(bool state)
{
  writeRegisterBit(MPU6050_REG_INT_ENABLE, 5, state);
}

bool MPU6050::getIntMotionEnabled(void)
{
  return readRegisterBit(MPU6050_REG_INT_ENABLE, 6);
}

void MPU6050::setIntMotionEnabled(bool state)
{
  writeRegisterBit(MPU6050_REG_INT_ENABLE, 6, state);
}

bool MPU6050::getIntFreeFallEnabled(void)
{
  return readRegisterBit(MPU6050_REG_INT_ENABLE, 7);
```

```
}

void MPU6050::setIntFreeFallEnabled(bool state)
{
  writeRegisterBit(MPU6050_REG_INT_ENABLE, 7, state);
}

uint8_t MPU6050::getMotionDetectionThreshold(void)
{
  return readRegister8(MPU6050_REG_MOT_THRESHOLD);
}

void MPU6050::setMotionDetectionThreshold(uint8_t threshold)
{
  writeRegister8(MPU6050_REG_MOT_THRESHOLD, threshold);
}

uint8_t MPU6050::getMotionDetectionDuration(void)
{
  return readRegister8(MPU6050_REG_MOT_DURATION);
}

void MPU6050::setMotionDetectionDuration(uint8_t duration)
{
  writeRegister8(MPU6050_REG_MOT_DURATION, duration);
}

uint8_t MPU6050::getZeroMotionDetectionThreshold(void)
{
  return readRegister8(MPU6050_REG_ZMOT_THRESHOLD);
}

void MPU6050::setZeroMotionDetectionThreshold(uint8_t threshold)
{
  writeRegister8(MPU6050_REG_ZMOT_THRESHOLD, threshold);
}

uint8_t MPU6050::getZeroMotionDetectionDuration(void)
{
  return readRegister8(MPU6050_REG_ZMOT_DURATION);
}

void MPU6050::setZeroMotionDetectionDuration(uint8_t duration)
{
  writeRegister8(MPU6050_REG_ZMOT_DURATION, duration);
}

uint8_t MPU6050::getFreeFallDetectionThreshold(void)
{
  return readRegister8(MPU6050_REG_FF_THRESHOLD);
}

void MPU6050::setFreeFallDetectionThreshold(uint8_t threshold)
{
```

```
  writeRegister8(MPU6050_REG_FF_THRESHOLD, threshold);
}

uint8_t MPU6050::getFreeFallDetectionDuration(void)
{
  return readRegister8(MPU6050_REG_FF_DURATION);
}

void MPU6050::setFreeFallDetectionDuration(uint8_t duration)
{
  writeRegister8(MPU6050_REG_FF_DURATION, duration);
}

bool MPU6050::getI2CMasterModeEnabled(void)
{
  return readRegisterBit(MPU6050_REG_USER_CTRL, 5);
}

void MPU6050::setI2CMasterModeEnabled(bool state)
{
  writeRegisterBit(MPU6050_REG_USER_CTRL, 5, state);
}

void MPU6050::setI2CBypassEnabled(bool state)
{
  return writeRegisterBit(MPU6050_REG_INT_PIN_CFG, 1, state);
}

bool MPU6050::getI2CBypassEnabled(void)
{
  return readRegisterBit(MPU6050_REG_INT_PIN_CFG, 1);
}

void MPU6050::setAccelPowerOnDelay(mpu6050_onDelay_t delay)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_MOT_DETECT_CTRL);
  value &= 0b11001111;
  value |= (delay << 4);
  writeRegister8(MPU6050_REG_MOT_DETECT_CTRL, value);
}

mpu6050_onDelay_t MPU6050::getAccelPowerOnDelay(void)
{
  uint8_t value;
  value = readRegister8(MPU6050_REG_MOT_DETECT_CTRL);
  value &= 0b00110000;
  return (mpu6050_onDelay_t)(value >> 4);
}

uint8_t MPU6050::getIntStatus(void)
{
  return readRegister8(MPU6050_REG_INT_STATUS);
}
```

```
Activites MPU6050::readActivites(void)
{
  uint8_t data = readRegister8(MPU6050_REG_INT_STATUS);

  a.isOverflow = ((data >> 4) & 1);
  a.isFreeFall = ((data >> 7) & 1);
  a.isInactivity = ((data >> 5) & 1);
  a.isActivity = ((data >> 6) & 1);
  a.isDataReady = ((data >> 0) & 1);

  data = readRegister8(MPU6050_REG_MOT_DETECT_STATUS);

  a.isNegActivityOnX = ((data >> 7) & 1);
  a.isPosActivityOnX = ((data >> 6) & 1);

  a.isNegActivityOnY = ((data >> 5) & 1);
  a.isPosActivityOnY = ((data >> 4) & 1);

  a.isNegActivityOnZ = ((data >> 3) & 1);
  a.isPosActivityOnZ = ((data >> 2) & 1);

  return a;
}

Vector MPU6050::readRawAccel(void)
{
  Wire.beginTransmission(mpuAddress);
#if ARDUINO >= 100
  Wire.write(MPU6050_REG_ACCEL_XOUT_H);
#else
  Wire.send(MPU6050_REG_ACCEL_XOUT_H);
#endif
  Wire.endTransmission();

  Wire.beginTransmission(mpuAddress);
  Wire.requestFrom(mpuAddress, 6);

  while (Wire.available() < 6);

#if ARDUINO >= 100
  uint8_t xha = Wire.read();
  uint8_t xla = Wire.read();
  uint8_t yha = Wire.read();
  uint8_t yla = Wire.read();
  uint8_t zha = Wire.read();
  uint8_t zla = Wire.read();
#else
  uint8_t xha = Wire.receive();
  uint8_t xla = Wire.receive();
  uint8_t yha = Wire.receive();
  uint8_t yla = Wire.receive();
  uint8_t zha = Wire.receive();
  uint8_t zla = Wire.receive();
```

```
#endif

  ra.XAxis = xha << 8 | xla;
  ra.YAxis = yha << 8 | yla;
  ra.ZAxis = zha << 8 | zla;

  return ra;
}

Vector MPU6050::readNormalizeAccel(void)
{
  readRawAccel();

  na.XAxis = ra.XAxis * rangePerDigit * 9.80665f;
  na.YAxis = ra.YAxis * rangePerDigit * 9.80665f;
  na.ZAxis = ra.ZAxis * rangePerDigit * 9.80665f;

  return na;
}

Vector MPU6050::readScaledAccel(void)
{
  readRawAccel();

  na.XAxis = ra.XAxis * rangePerDigit;
  na.YAxis = ra.YAxis * rangePerDigit;
  na.ZAxis = ra.ZAxis * rangePerDigit;

  return na;
}


Vector MPU6050::readRawGyro(void)
{
  Wire.beginTransmission(mpuAddress);
#if ARDUINO >= 100
  Wire.write(MPU6050_REG_GYRO_XOUT_H);
#else
  Wire.send(MPU6050_REG_GYRO_XOUT_H);
#endif
  Wire.endTransmission();

  Wire.beginTransmission(mpuAddress);
  Wire.requestFrom(mpuAddress, 6);

  while (Wire.available() < 6);

#if ARDUINO >= 100
  uint8_t xha = Wire.read();
  uint8_t xla = Wire.read();
  uint8_t yha = Wire.read();
  uint8_t yla = Wire.read();
  uint8_t zha = Wire.read();
  uint8_t zla = Wire.read();
```

```
#else
  uint8_t xha = Wire.receive();
  uint8_t xla = Wire.receive();
  uint8_t yha = Wire.receive();
  uint8_t yla = Wire.receive();
  uint8_t zha = Wire.receive();
  uint8_t zla = Wire.receive();
#endif

  rg.XAxis = xha << 8 | xla;
  rg.YAxis = yha << 8 | yla;
  rg.ZAxis = zha << 8 | zla;

  return rg;
}

Vector MPU6050::readNormalizeGyro(void)
{
  readRawGyro();

  if (useCalibrate)
  {
    ng.XAxis = (rg.XAxis - dg.XAxis) * dpsPerDigit;
    ng.YAxis = (rg.YAxis - dg.YAxis) * dpsPerDigit;
    ng.ZAxis = (rg.ZAxis - dg.ZAxis) * dpsPerDigit;
  } else
  {
    ng.XAxis = rg.XAxis * dpsPerDigit;
    ng.YAxis = rg.YAxis * dpsPerDigit;
    ng.ZAxis = rg.ZAxis * dpsPerDigit;
  }

  if (actualThreshold)
  {
    if (abs(ng.XAxis) < tg.XAxis) ng.XAxis = 0;
    if (abs(ng.YAxis) < tg.YAxis) ng.YAxis = 0;
    if (abs(ng.ZAxis) < tg.ZAxis) ng.ZAxis = 0;
  }

  return ng;
}

float MPU6050::readTemperature(void)
{
  int16_t T;
  T = readRegister16(MPU6050_REG_TEMP_OUT_H);
  return (float)T / 340 + 36.53;
}

int16_t MPU6050::getGyroOffsetX(void)
{
  return readRegister16(MPU6050_REG_GYRO_XOFFS_H);
}
```

```
int16_t MPU6050::getGyroOffsetY(void)
{
  return readRegister16(MPU6050_REG_GYRO_YOFFS_H);
}

int16_t MPU6050::getGyroOffsetZ(void)
{
  return readRegister16(MPU6050_REG_GYRO_ZOFFS_H);
}

void MPU6050::setGyroOffsetX(int16_t offset)
{
  writeRegister16(MPU6050_REG_GYRO_XOFFS_H, offset);
}

void MPU6050::setGyroOffsetY(int16_t offset)
{
  writeRegister16(MPU6050_REG_GYRO_YOFFS_H, offset);
}

void MPU6050::setGyroOffsetZ(int16_t offset)
{
  writeRegister16(MPU6050_REG_GYRO_ZOFFS_H, offset);
}

int16_t MPU6050::getAccelOffsetX(void)
{
  return readRegister16(MPU6050_REG_ACCEL_XOFFS_H);
}

int16_t MPU6050::getAccelOffsetY(void)
{
  return readRegister16(MPU6050_REG_ACCEL_YOFFS_H);
}

int16_t MPU6050::getAccelOffsetZ(void)
{
  return readRegister16(MPU6050_REG_ACCEL_ZOFFS_H);
}

void MPU6050::setAccelOffsetX(int16_t offset)
{
  writeRegister16(MPU6050_REG_ACCEL_XOFFS_H, offset);
}

void MPU6050::setAccelOffsetY(int16_t offset)
{
  writeRegister16(MPU6050_REG_ACCEL_YOFFS_H, offset);
}

void MPU6050::setAccelOffsetZ(int16_t offset)
{
  writeRegister16(MPU6050_REG_ACCEL_ZOFFS_H, offset);
}
```

```
// Calibrate algorithm
void MPU6050::calibrateGyro(uint8_t samples)
{
  // Set calibrate
  useCalibrate = true;

  // Reset values
  float sumX = 0;
  float sumY = 0;
  float sumZ = 0;
  float sigmaX = 0;
  float sigmaY = 0;
  float sigmaZ = 0;

  // Read n-samples
  for (uint8_t i = 0; i < samples; ++i)
  {
    readRawGyro();
    sumX += rg.XAxis;
    sumY += rg.YAxis;
    sumZ += rg.ZAxis;

    sigmaX += rg.XAxis * rg.XAxis;
    sigmaY += rg.YAxis * rg.YAxis;
    sigmaZ += rg.ZAxis * rg.ZAxis;

    delay(5);
  }

  // Calculate delta vectors
  dg.XAxis = sumX / samples;
  dg.YAxis = sumY / samples;
  dg.ZAxis = sumZ / samples;

  // Calculate threshold vectors
  th.XAxis = sqrt((sigmaX / 50) - (dg.XAxis * dg.XAxis));
  th.YAxis = sqrt((sigmaY / 50) - (dg.YAxis * dg.YAxis));
  th.ZAxis = sqrt((sigmaZ / 50) - (dg.ZAxis * dg.ZAxis));

  // If already set threshold, recalculate threshold vectors
  if (actualThreshold > 0)
  {
    setThreshold(actualThreshold);
  }
}

// Get current threshold value
uint8_t MPU6050::getThreshold(void)
{
  return actualThreshold;
}

// Set treshold value
```

```
void MPU6050::setThreshold(uint8_t multiple)
{
  if (multiple > 0)
  {
    // If not calibrated, need calibrate
    if (!useCalibrate)
    {
      calibrateGyro();
    }

    // Calculate threshold vectors
    tg.XAxis = th.XAxis * multiple;
    tg.YAxis = th.YAxis * multiple;
    tg.ZAxis = th.ZAxis * multiple;
  } else
  {
    // No threshold
    tg.XAxis = 0;
    tg.YAxis = 0;
    tg.ZAxis = 0;
  }

  // Remember old threshold value
  actualThreshold = multiple;
}

// Fast read 8-bit from register
uint8_t MPU6050::fastRegister8(uint8_t reg)
{
  uint8_t value;

  Wire.beginTransmission(mpuAddress);
#if ARDUINO >= 100
  Wire.write(reg);
#else
  Wire.send(reg);
#endif
  Wire.endTransmission();

  Wire.beginTransmission(mpuAddress);
  Wire.requestFrom(mpuAddress, 1);
#if ARDUINO >= 100
  value = Wire.read();
#else
  value = Wire.receive();
#endif
  Wire.endTransmission();

  return value;
}

// Read 8-bit from register
uint8_t MPU6050::readRegister8(uint8_t reg)
{
```

```
  uint8_t value;

  Wire.beginTransmission(mpuAddress);
#if ARDUINO >= 100
  Wire.write(reg);
#else
  Wire.send(reg);
#endif
  Wire.endTransmission();

  Wire.beginTransmission(mpuAddress);
  Wire.requestFrom(mpuAddress, 1);
  while (!Wire.available()) {};
#if ARDUINO >= 100
  value = Wire.read();
#else
  value = Wire.receive();
#endif
  Wire.endTransmission();

  return value;
}

// Write 8-bit to register
void MPU6050::writeRegister8(uint8_t reg, uint8_t value)
{
  Wire.beginTransmission(mpuAddress);

#if ARDUINO >= 100
  Wire.write(reg);
  Wire.write(value);
#else
  Wire.send(reg);
  Wire.send(value);
#endif
  Wire.endTransmission();
}

int16_t MPU6050::readRegister16(uint8_t reg)
{
  int16_t value;
  Wire.beginTransmission(mpuAddress);
#if ARDUINO >= 100
  Wire.write(reg);
#else
  Wire.send(reg);
#endif
  Wire.endTransmission();

  Wire.beginTransmission(mpuAddress);
  Wire.requestFrom(mpuAddress, 2);
  while (!Wire.available()) {};
#if ARDUINO >= 100
  uint8_t vha = Wire.read();
```

```
  uint8_t vla = Wire.read();
#else
  uint8_t vha = Wire.receive();
  uint8_t vla = Wire.receive();
#endif
  Wire.endTransmission();

  value = vha << 8 | vla;

  return value;
}

void MPU6050::writeRegister16(uint8_t reg, int16_t value)
{
  Wire.beginTransmission(mpuAddress);

#if ARDUINO >= 100
  Wire.write(reg);
  Wire.write((uint8_t)(value >> 8));
  Wire.write((uint8_t)value);
#else
  Wire.send(reg);
  Wire.send((uint8_t)(value >> 8));
  Wire.send((uint8_t)value);
#endif
  Wire.endTransmission();
}

// Read register bit
bool MPU6050::readRegisterBit(uint8_t reg, uint8_t pos)
{
  uint8_t value;
  value = readRegister8(reg);
  return ((value >> pos) & 1);
}

// Write register bit
void MPU6050::writeRegisterBit(uint8_t reg, uint8_t pos, bool state)
{
  uint8_t value;
  value = readRegister8(reg);

  if (state)
  {
    value |= (1 << pos);
  } else
  {
    value &= ~(1 << pos);
  }

  writeRegister8(reg, value);
}
```

PIC32 with MPU-6050

```
#include <p32xxxx.h>
#include <proc/p32mx795f512l.h>
#include "LCD.h"
#include <stdio.h>

char Data[10];

char direction;
// forward: 'F'
// backward: 'B'
// clockwise: 'C'
// counterclockwise: 'U'
// stop 'S'

void SetDMA() {
    IEC1CLR = 0x00010000;
    IFS1CLR = 0x00010000;
    DMACONSET = 0x00008000;
    DCH0CON = 0x03;
    DCH0ECON = (27 << 8) | 0x10;
    DCH0SSA = ((int) (&U1RXREG)) & 0x1FFFFFFF;
    DCH0DSA = ((int) (Data)) & 0x1FFFFFFF;
    DCH0SSIZ = 1;
    DCH0DSIZ = 10;
    DCH0CSIZ = 1; //one byte per UART transfer request
    DCH0INTCLR = 0x00ff00ff; //clear existing events,
    DCH0INTbits.CHBCIE = 1;
    // DCH0INTSET = 0x00090000; //enable Block Complete and error intr
    IPC9bits.DMA0IP = 0b101; //set interrupt priority to 5,
    IPC9bits.DMA0IS = 0b010; //set interrupt sub-priority to 2
    IEC1SET = 0x00010000; //enable DMA channel 0 interrupt
    asm("di");
    IPC9bits.DMA0IP = 3;
    IFS1bits.DMA0IF = 0;
    IEC1bits.DMA0IE = 1;
    asm("ei");
    DCH0CONSET = 0x80; //turn on channel 0
}
#pragma interrupt DMA_C0_ISR ipl3 vector 36

void DMA_C0_ISR() {
    int forward = 0;
    int backward = 0;
    int clockwise = 0;
    int counterclockwise = 0;
    int stop = 0;
    for (int i = 0; i < 1; i++) {
        if (Data[i] == 'F')
            forward++;
```

```
        else if (Data[i] == 'B')
            backward++;
        else if (Data[i] == 'C')
            clockwise++;
        else if (Data[i] == 'U')
            counterclockwise++;
        else stop++;
        Data[i] = 'S';
    }
    direction = 'S';
    int curr_max = stop;
    if (curr_max < forward) {
        direction = 'F';
        curr_max = forward;
    }
    if (curr_max < backward) {
        direction = 'B';
        curr_max = backward;
    }
    if (curr_max < clockwise) {
        direction = 'C';
        curr_max = clockwise;
    }
    if (curr_max < counterclockwise)
        direction = 'U';
    U1TXREG = direction;
    SetDMA();
}

void SetUART1(unsigned int baud_setting) {
    U1BRG = baud_setting;
    U1MODE = 0; // default all zero
    U1STA = 0;
    IFS0bits.U1RXIF = 0;
    U1STAbits.URXEN = 1;
    U1STAbits.UTXEN = 1;
    U1MODEbits.ON = 1;
}

void main() {
    direction = 'S';
    for (int i = 0; i < 10; i++)
        Data[i] = 'S';
    MCU_init();
    LCD_init();
    SetUART1(25);
    LCD_puts("ON");
    SetDMA();
    while (1);
}
```

LCD

```
#ifndef LCD_H
#define LCD_H
#include <p32xxxx.h>
#define LCD_IDLE 0x33
#define LCD_2_LINE_4_BITS 0x28
#define LCD_2_LINE_8_BITS 0x38
#define LCD_DSP_CSR 0x0c
#define LCD_CLR_DSP 0x01
#define LCD_CSR_INC 0x06
#define LCD_SFT_MOV 0x14
/* define macros for interfacing ports */
#define RS PORTDbits.RD0
#define E PORTDbits.RD1
typedef unsigned char uchar;
/* define constant strings for display */
/* Function prototypes */
void MCU_init(void);
void LCD_init(void);
void LCD_putchar(uchar c);
void LCD_puts(const uchar *s);
void LCD_goto(uchar addr);
void GenMsec(void);
void DelayUsec(uchar num);
void DelayMsec(uchar num);

/* initialize the PIC32 MCU */
void MCU_init() {
    /* setup I/O ports to connect to the LCD module */
    SYSKEY = 0x0; //write invalid key to force lock
    SYSKEY = 0xAA996655; //write Key1 to SYSKEY
    SYSKEY = 0x556699AA; //Write Key2 to SYSKEY
    OSCCONbits.PBDIV = 0b00; //PBCLK = SYSCLK/8
    SYSKEY = 0x0; //Write invalid key to force lock
    TRISE = 0x0000;
    PORTE = 0x0000;
    TRISDbits.TRISD0 = 0;
    TRISDbits.TRISD1 = 0;
    PORTDbits.RD0 = 0;
    PORTDbits.RD1 = 0;
    /* setup Timer and interrupt */
    /* PR1 is not set */
    INTCONbits.MVEC = 1;
    asm("ei");
    T1CON = 0;
    IPC1bits.T1IP = 4;
    IFS0bits.T1IF = 0;
    IEC0bits.T1IE = 1;
    TMR1 = 0;
    T1CONbits.TCKPS = 0b00;
    T1CONbits.TCS = 0;
    T1CONbits.TGATE = 0;
}
```

```
/* initialize the LCD module */
void LCD_init() {
    DelayMsec(15); //wait for 15 ms
    RS = 0; //send command
    PORTE = LCD_IDLE; //function set - 8 bit interface
    DelayMsec(5); //wait for 5 ms
    PORTE = LCD_IDLE; //function set - 8 bit interface
    DelayUsec(100); //wait for 100 us
    PORTE = LCD_IDLE; //function set
    DelayMsec(5);
    PORTE = LCD_IDLE;
    DelayUsec(100);
    LCD_putchar(LCD_2_LINE_4_BITS);
    DelayUsec(40);
    LCD_putchar(LCD_DSP_CSR);
    DelayUsec(40);
    LCD_putchar(LCD_CLR_DSP);
    DelayMsec(5);
    LCD_putchar(LCD_CSR_INC);
}

/* Send one byte c (instruction or data) to the LCD */
void LCD_putchar(uchar c) {
    E = 1;
    PORTE = c; //sending higher nibble
    E = 0; //producing falling edge on E
    E = 1;
    PORTE <<= 4; //sending lower nibble through higher 4 ports
    E = 0; //producing falling edge on E
}

/* Display a string of characters *s by continuously calling
↪  LCD_putchar() */
void LCD_puts(const uchar *s) {
    RS = 1;
    int pos = 0;
    while (s[pos] != '\0') {
        LCD_putchar(s[pos++]);
        DelayUsec(40);
    }
    RS = 0;
}

/* go to a specific DDRAM address addr */
void LCD_goto(uchar addr) {
    LCD_putchar(addr | 0b10000000);
    DelayUsec(40);
}

/* configure timer SFRs to generate num us delay*/
void DelayUsec(uchar num) {
    PR1 = (num << 2);
    TMR1 = 0x0000;
```

```
    T1CONbits.ON = 1;
    while (T1CONbits.ON);
}

/* configure timer SFRs to generate 1 ms delay*/
void GenMsec() {
    PR1 = 4000;
    TMR1 = 0x0000;
    T1CONbits.ON = 1;
    while (T1CONbits.ON);
}

/* Call GenMsec() num times to generate num ms delay*/
void DelayMsec(uchar num) {
    uchar i;
    for (i = 0; i < num; i++)
        GenMsec();
}
#pragma interrupt T1_ISR ipl4 vector 4

void T1_ISR(void) {
    T1CONbits.ON = 0;
    IFS0bits.T1IF = 0;
}
#endif
```

Code for the Robot

PIC32 with Servos

```
#include <p32xxxx.h>
#include <proc/p32mx795f512l.h>
#pragma interrupt T1_ISR ipl5 vector 4

void T1_ISR(void) {
    T1CONbits.ON = 0;
    IFS0bits.T1IF = 0;
}

void GenMsec() {
    PR1 = 4000;
    TMR1 = 0x0000;
    T1CONbits.ON = 1;
    while (T1CONbits.ON);
}

void DelayMsec(int num) {
    int i;
    for (i = 0; i < num; i++) {
        GenMsec();
    }
```

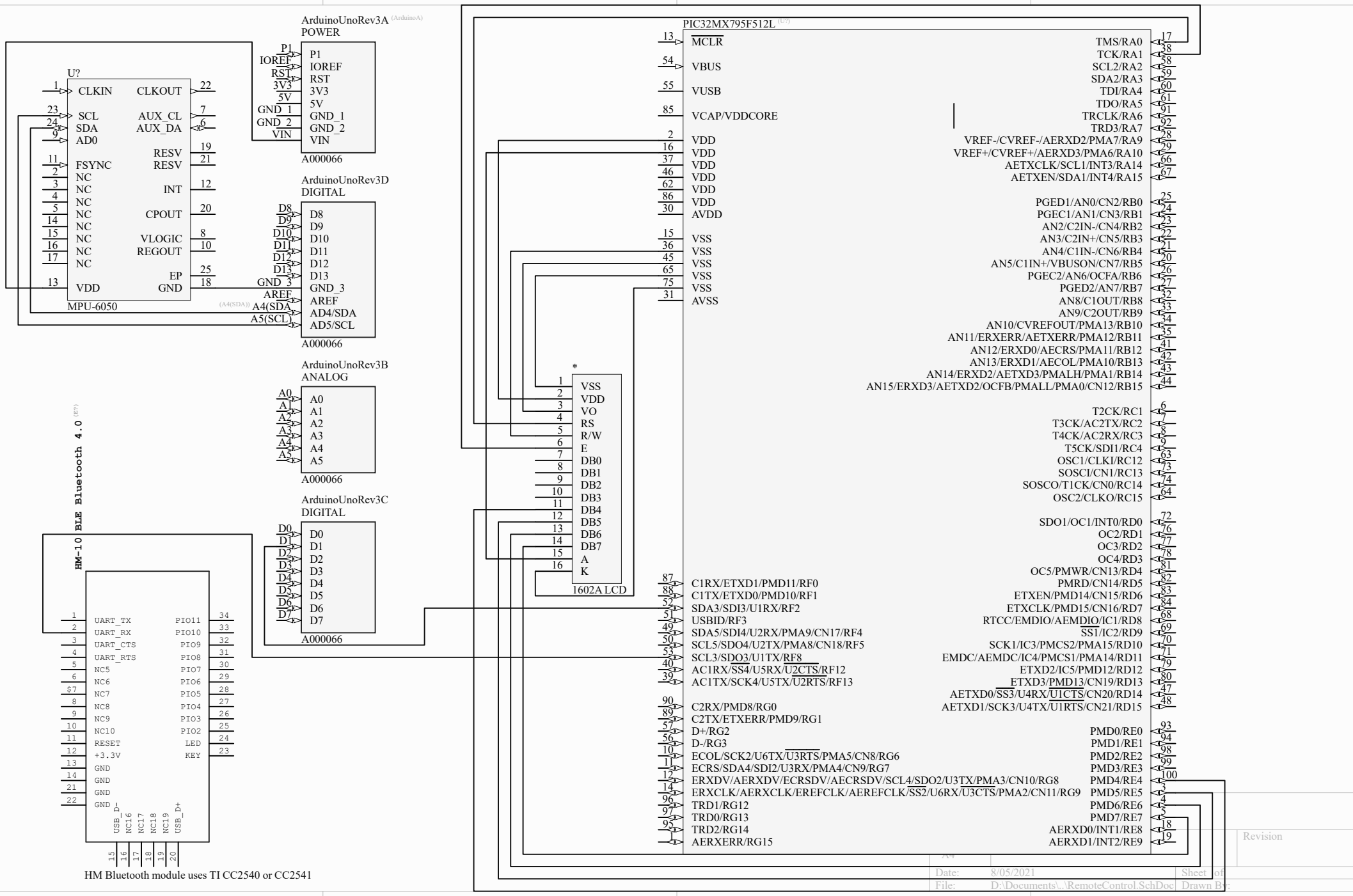
```
}

void DelayUsec(int num) {
    PR1 = (num << 2);
    TMR1 = 0x0000;
    T1CONbits.ON = 1;
    while (T1CONbits.ON);
}

char CheckObs() {
    TRISDbits.TRISD10 = 1;
    PR3 = 59999; // period = 60ms
    T3CONbits.TCKPS = 0b011; // 1:8
    TMR3 = 0;
    int start = 0;
    int finish = 0;
    T3CONbits.ON = 1;
    TRISBbits.TRISB1 = 0;
    PORTBbits.RB1 = 1;
    DelayUsec(10);
    PORTBbits.RB1 = 0;
    while (PORTDbits.RD10 == 0 && TMR3 < 55000);
    if (TMR3 < 55000) start = TMR3;
    while (PORTDbits.RD10 == 1 && TMR3 < 55000);
    if (TMR3 < 55000) finish = TMR3;
    while (TMR3 < 55000);
    T3CONbits.ON = 0;
    if (start != 0 && finish != 0 && (finish - start) < 2000)
        return 'N';
    return 'Y';
}

void Init() {
    SYSKEY = 0x0; //write invalid key to force lock
    SYSKEY = 0xAA996655; //write Key1 to SYSKEY
    SYSKEY = 0x556699AA; //Write Key2 to SYSKEY
    OSCCONbits.PBDIV = 0b00;
    SYSKEY = 0x0; //Write invalid key to force lock

    U1BRG = 25;
    U1MODE = 0;
    U1STA = 0;
    IFS0bits.U1RXIF = 0;
    U1STAbits.URXEN = 1;
    U1MODEbits.ON = 1;

    asm("di");
    INTCONbits.MVEC = 1;
    T1CON = 0; // Reset Timer1 control
    IPC1bits.T1IP = 5; // Timer1 Group Priority = 2
    IFS0bits.T1IF = 0; // Reset Interrupt flag
    IEC0bits.T1IE = 1; // Enable Timer1 Individual Interrupt
    TMR1 = 0; // Reset TMR1
    T1CONbits.TCKPS = 0b00; // T1CLK = PBCLK = 4Mhz
```

```
    T1CONbits.TCS = 0; // Source: PBCLK
    T1CONbits.TGATE = 0; // Disable gated mode
    asm("ei");

    PR2 = 9999; // PWM signal period = 0.01s
    T2CONbits.TCKPS = 0b011; // 1:8
    T2CONSET = 0x8000; // start Timer 2

    OC1CON = 0x0000; // stop OC1 module
    OC1RS = 750; // initialize duty cycle register
    OC1R = 750; // initialize OC1R register for the first time
    OC1CON = 0x0006; // OC1 16-bit, Timer 2, in PWM mode w/o FP
    OC1CONSET = 0x8000;

    OC2CON = 0x0000;
    OC2RS = 750;
    OC2R = 750;
    OC2CON = 0x0006;
    OC2CONSET = 0x8000;

    OC3CON = 0x0000;
    OC3RS = 750;
    OC3R = 750;
    OC3CON = 0x0006;
    OC3CONSET = 0x8000;

    OC4CON = 0x0000;
    OC4RS = 750;
    OC4R = 750;
    OC4CON = 0x0006;
    OC4CONSET = 0x8000;

    DelayMsec(500);
}

void left() {
    OC1RS = 900;
    DelayMsec(200);
    OC2RS = 900;
    DelayMsec(200);
    OC1RS = 750;
    OC3RS = 550;
    DelayMsec(200);
    OC2RS = 750;
    DelayMsec(200);
    OC3RS = 750;
    DelayMsec(200);
}

void right() {
    OC3RS = 550;
    DelayMsec(200);
    OC4RS = 600;
    DelayMsec(200);
```

```
    OC3RS = 750;
    OC1RS = 900;
    DelayMsec(200);
    OC4RS = 750;
    DelayMsec(200);
    OC1RS = 750;
    DelayMsec(200);
}

void stop() {
    OC1RS = 750;
    OC2RS = 750;
    OC3RS = 750;
    OC4RS = 750;
    DelayMsec(200);
}

void forward() {
    if (CheckObs() == 'N') {
        stop();
        return;
    }
    OC3RS = 550;
    for (int i = 0; i < 10; i++) {
        OC2RS += 20;
        DelayMsec(50);
    }
    DelayMsec(200);
    OC1RS = 1000;
    DelayMsec(200);
    OC2RS = 750;
    DelayMsec(200);
    OC1RS = 750;
    OC3RS = 750;
    DelayMsec(200);

    OC1RS = 900;
    for (int i = 0; i < 10; i++) {
        OC4RS -= 20;
        DelayMsec(50);
    }
    DelayMsec(200);
    OC3RS = 550;
    DelayMsec(200);
    OC4RS = 750;
    DelayMsec(200);
    OC3RS = 750;
    OC1RS = 750;
    DelayMsec(200);
}

void backward() {
    OC3RS = 550;
    DelayMsec(200);
```

```
    OC4RS = 550;
    DelayMsec(200);
    OC3RS = 750;
    OC1RS = 1000;
    DelayMsec(200);
    for (int i = 0; i < 10; i++) {
        OC4RS += 20;
        DelayMsec(50);
    }
    DelayMsec(200);
    OC1RS = 750;
    DelayMsec(200);

    OC1RS = 950;
    DelayMsec(200);
    OC2RS = 900;
    DelayMsec(200);
    OC1RS = 750;
    OC3RS = 500;
    DelayMsec(200);
    for (int i = 0; i < 10; i++) {
        OC2RS -= 15;
        DelayMsec(50);
    }
    DelayMsec(200);
    OC3RS = 750;
    DelayMsec(200);
}

void main() {
    char dir = 'S';
    Init();
    while (1) {
        while (U1STAbits.URXDA)
            dir = U1RXREG;
        if (dir == 'C') right();
        if (dir == 'U') left();
        if (dir == 'F') forward();
        if (dir == 'B') backward();
        else stop();
    }
}
```

Schematics (Original Print)


ArduinoUnoRev3A
POWER
A000066
ArduinoUnoRev3D
DIGITAL
A000066
ArduinoUnoRev3B
ANALOG
A000066
ArduinoUnoRev3C
DIGITAL
A000066
U?
MPU-6050
HM-10 BLE Bluetooth 4.0
HM Bluetooth module uses TI CC2540 or CC2541
1602A LCD
PIC32MX795F512L
Date: 8/05/2021
File: D:\Documents\..\RemoteControl.SchDoc
Sheet of
Drawn By:
Revision