**DANANG UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**CENTER OF EXCELLENCE**



**CAPSTONE PROJECT REPORT**

**M-2 REPORT**

**SIMULATE AUTO-FIND TARGET CAR**

Students: Bui Tuan Anh 11ES

Nguyen Hoang Anh Khoa 11ES

Nguyen Truong Trong Nhan 11ES

Tran Ngoc Quang 11ES

Nguyen Tin 11ES

Supervisor: Pham Minh Hai FPT

Dr. Nguyen Hoang Mai DUT

Dr. Truong Thi Bich Thanh DUT

Contents

[**I. Introduction** 3](#_Toc450915956)

[**II. High-Level Design** 3](#_Toc450915957)

[1. Rationale of Project Idea 3](#_Toc450915958)

[Logical Structure 3](#_Toc450915959)

[Motion planning with Kinematics and Dynamics 5](#_Toc450915960)

[**III. Hardware Design** 7](#_Toc450915961)

[Hardware Details: 7](#_Toc450915962)

[1. Remote Control Car 7](#_Toc450915963)

[2. Ultrasonic Sensor 8](#_Toc450915964)

[3. ESC Circuit 15](#_Toc450915965)

[4. Color Sensor TCS3200 17](#_Toc450915966)

[5. MPU-6050 Six-Axis (Gyro + Accelerometer) 18](#_Toc450915967)

[6. Power Supply 19](#_Toc450915968)

[7. CAN Module (Controller Area Network) 21](#_Toc450915969)

[8. MCU 22](#_Toc450915970)

[Hardware Issues 23](#_Toc450915971)

[**IV. Software Design** 25](#_Toc450915972)

[1. Obstacles Avoidance Algorithm 25](#_Toc450915973)

[2. Auto Find Target Algorithm 26](#_Toc450915974)

[3. MPU-6050 27](#_Toc450915975)

[4. Ultrasonic Sensor 29](#_Toc450915976)

[5. CAN Controller 31](#_Toc450915977)

[6. Arduino Timers and Interrupts 35](#_Toc450915978)

[**V. References** 37](#_Toc450915979)

[Datasheets 39](#_Toc450915980)

[Useful Websites 39](#_Toc450915981)

"A modified remote control car that drives autonomously between lanes and avoids forward collisions."

**I. Introduction**

For our final project, we re-engineered a remote control car to autonomously navigate through a road by detecting lanes and centering itself between them as well as detect objects in front of it and avoid collision. The RC car detects lanes through color input from a color sensor mounted at its front. Using an ultrasonic distance sensor, the car determines when to stop accelerating once a certain distance between a forward object has been breached. All computations based on sensor data are handled by an Atmega328p MCU. Due to the nature of the input peripherals, this system is extremely time sensitive so that computations had to be optimized as much as possible in order for the car to be able to react and respond with proper movements in real time. In addition, given the limited computational capacity of this 32-bit MCU, our design made use of several computational efficiency strategies.

**II. High-Level Design**

## 1. Rationale of Project Idea

The basic idea of our project stemmed from a rather playful interest in using an MCU to program a robot to perform some common everyday function autonomously. Given our lack of mechanical engineering expertise, an RC car was chosen as the hardware to be programmed since it provides all the components and infrastructure necessary for a car mechanical system, all at a low cost with an easy-to-use interface. We decided to design the car so that it would automatically navigate a road made of parallel lines which were meant to mimic the roads that actual full-sized cars encounter. Given that there has been some societal interest in the development of automatic driving car technology, we figured that designing a low complexity system with this functionality would be an exciting and practical project to pursue.

## Logical Structure

Our high-level hardware design consists of seven main blocks as shown in the block diagram.



*Figure 01: Structure of project*

All computations and processing are handled by the Atmega328p MCU. The MCU receives inputs from two peripherals: the color sensor, and the ultrasonic distance sensor.

Color sensor input in the color of object in road such as, color of road lines, target and start points. Since our RC car only has to detect the color of a road line in contrast to the rest of its environment.

The voltage threshold for the analog comparator had to be adjusted to a proper value for a given road line color which we chose to be black. Processing of this color input from the color sensor is done by the MCU to generate signals that indicate whether the car should turn left or right or go straight. These command signals eventually control the servo motor on the RC car that allows it to turn left or right, but must first go through an H-Bridge circuit so that polar voltages can be created to drive the servo motor in both directions (since the MCU cannot generate negative voltages).

The MCU also receives a digital input from the distance sensor which is processed with the MCU's internal ADC into a 16-bit value. This digital value can be mapped to an actual distance, but is not done in software since it would be unnecessary and would only add more computation time to an already time sensitive system. The 32-bit digital distance value is used in a simple control algorithm that determines the duty cycle of the PWM coming out of Timer2. The PWM controls the operation of the car’s DC motor, supplying the motor with a higher voltage when its duty cycle is higher. The PWM is sent to a motor control circuit which uses an optoisolator to separate the MCU circuitry from the motor circuitry. The optoisolator is necessary so that excessive current draws from the motor do not damage the MCU hardware.

## Motion planning with Kinematics and Dynamics

Geometric constraints are generally not sufficient to adequately express robot motions. Constraints on velocity, forces, torques, and accelerations are needed for better representations. Implicit velocity constraints express velocities that are not allowed, and are of the form

s some function g : Q X \_Q 🡪 R

Can be any of the symbols =; <; >;;

Parametric velocity constraints express velocities that are allowed, and are of the form

\_ q = f (q; u)

Where f (q; u) is some function f: Q x U 🡪 \_Q that expresses a set of differential equations

U is an input control

Car configuration: q = (x; y; ) R x S1

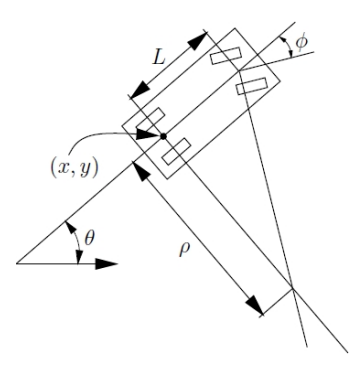
Body frame

* Origin is at the center of rear axle
* x -axis points along main axis of the car
* Velocity (signed speed): s
* Steering angle:
* Express car motions as a set of deferential equations

\_ x = f1(x; y;; s ; )

\_ y = f2(x; y;; s ; )

\_ = f3(x; y;; s ; )



*Figure 02: Kinematic dimension on car*

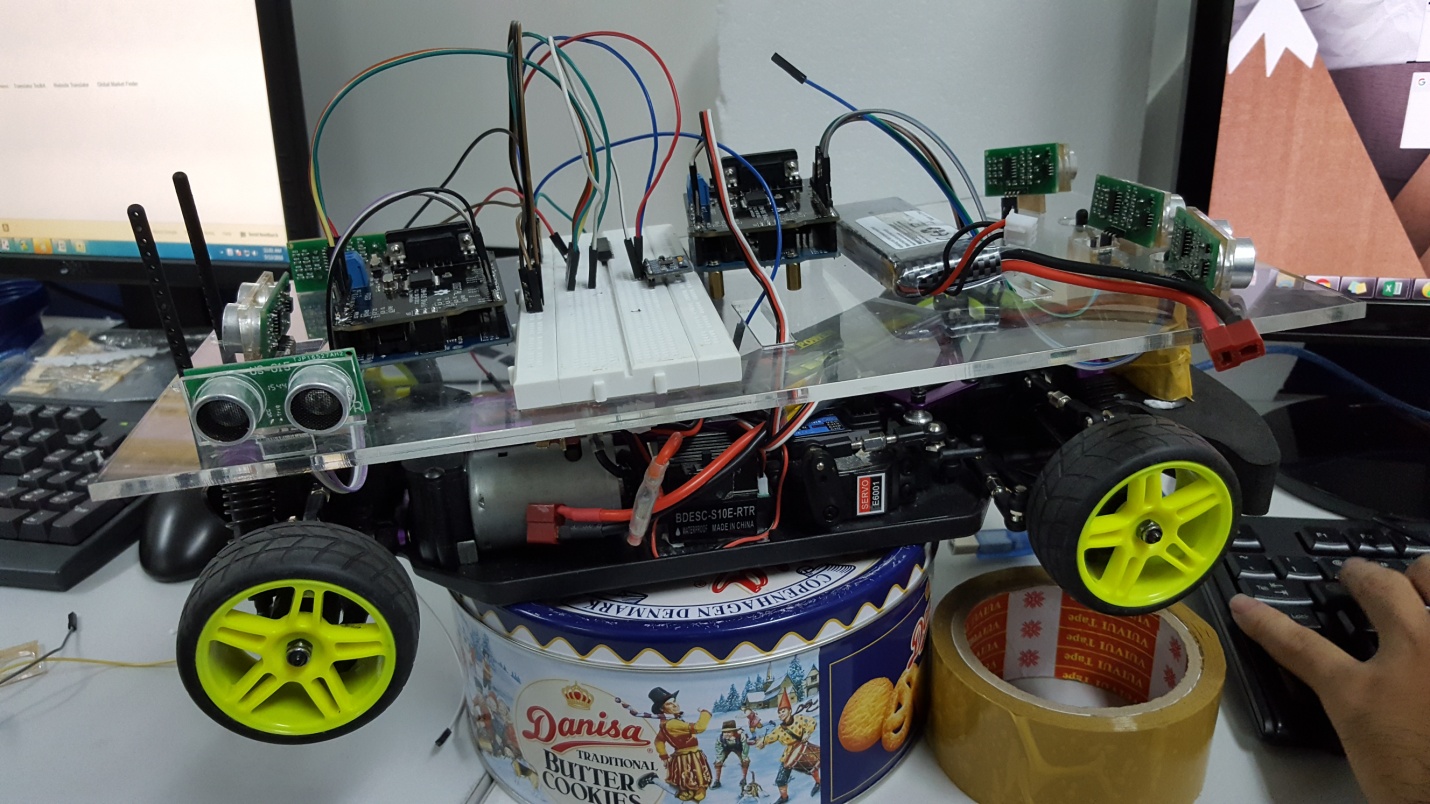
In a small time interval t, the car must move approximately in the direction that the rear wheels are pointing

In the limit, as t 🡪 0, the

* w : distance traveled by the car
* If the steering angle is fixed at , the car travels in circular motion, in which the radius of the circle is
* where L is the distance from front to rear axles

Therefore:

**III. Hardware Design**



*Figure 03: Demo production*

Due to the low cost and relatively low complexity of our design, this project can easily be rebuilt by those interested in tinkering around with an RC car. The following sections describe the hardware that is used in detail and explains how they are set up.

## Hardware Details:

### Remote Control Car

The RC car used in our project was the cheapest RC car. It is a 4-wheel car with rear-wheel drive and front-wheel steering. Since a datasheet for the hardware was not provided with it, we had to manually test the car’s connections and figure out how to properly operate its motor functions ourselves. Given the nature of our project, the RF functionality of the car was scrapped and the receiver board on the car was removed. Only seven wires had to be tested to determine their function - two wires for the car battery, two wires for the rear-wheel DC motor, and three wires for the front-wheel servo motor.

The voltage across the car battery hovers around 7.5V, which was expected since the car requires five AA batteries rated at a nominal 1.5V each. To test the servo motor wires, we applied different voltages across it starting low and incrementing the voltage until a turning response by the motor was achieved. The servo motor turns in one direction with a positive voltage across it, and turns the other direction with a negative voltage across it, but does not have the capability to turn at varying angles. The servo motor begins responding at around a 3.8V threshold but reaches the max turn angle at a much slower rate. Applying a larger voltage across the motor causes the motor to reach the set turn angle faster.

Measurements were taken to determine the effect of varying voltages across the DC motor. We were able to get a relatively accurate measurement of the car’s wheel RPM for a given DC voltage across the motor. The data gathered shows us that the threshold voltage for the DC motor to power actual motion of the wheels is around 1.3V and that the voltage-to-RPM relationship is linear.

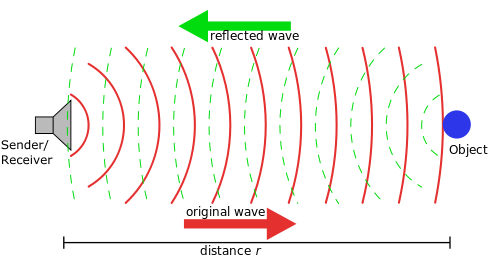
### Ultrasonic Sensor

There are various types of ultrasonic sensors, based on use such as recognizing in different distance, recognizing things with

**Ultrasonic sensor and the TOF (Time of Flight) rule:**

Time-of-flight is the principle mode of operation for most radar, laser and active acoustic devices. This technique uses the time between the transmission of a pulse and the reception of an echo is measured to provide range.

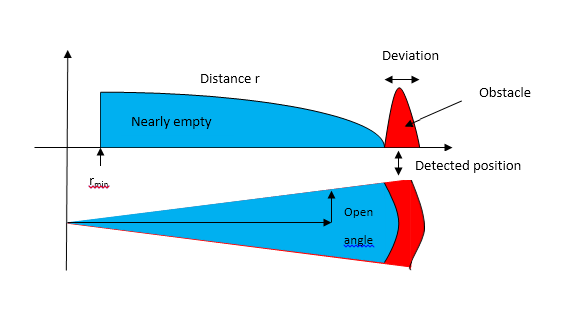
Because the round-trip time is equal two times the distance between the sensor and the obstacle, following the transmitted wave’s direction. Or the distance between the sensor and the obstacle is will be measure by using the following formula:



*Figure 04: Illusion of ultrasonic sensor’s working*

**Ultrasonic sensor range**

Ultrasonic sensor can be modeling as a sector, where the inner points seem to have no obstacle, and the points at the border seem to have obstacles.



*Figure 05: Area sensor working*

**Specification of some SRF ultrasonic sensor**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sensor** | **Information** | **Range** | | **Angle**  **\*** | **Echoes**  **\*\*** | **Different time** | **Note** |
| **Minimum** | **Maximum** |
| SRF02 | 12C / Serial | 15 cm | 6 m | 45­o | One | 70 ms | A |
| SRF04 | Digital | 3 cm | 3 cm | 45­o | One | 100 µs – 36 ms |  |
| SRF05 | Digital | 3 cm | 4 cm | 45­o | One | 100 µs – 36 ms |  |
| SRF08 | 12C | 3 cm | 6 cm | 45­o | 17 | 65 ms | BC |
| SRF10 | 12C | 3 cm | 6 cm | 60o | One | 65 ms | AB |
| SRF235 | 12C | 10 cm | 1,2 cm | 15o | One | 10 ms | AD |

*Table 01: Compare types of ultrasonic sensors;*

\* The estimated angle of the sensor sector

\*\* The echo recorded by sensor. These are the echo recorded from the latest read, and overwrite every different time.

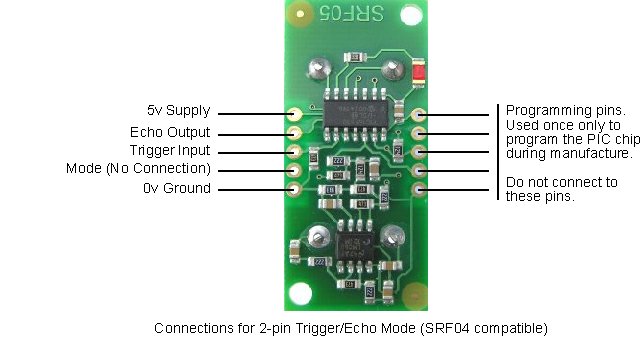
A: Sensors that have smaller size than typical (SRF04/ 05/ 08).

B: Time range is adjustable.

C: This sensor also include a photocell at the front to detect light.

D: Work at a frequency higher than 235 kHz.

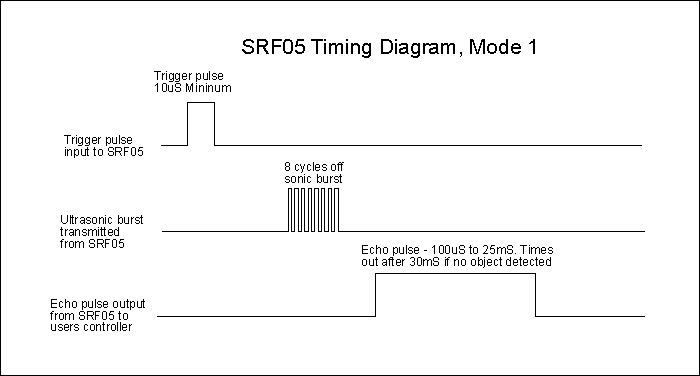
This mode uses separate trigger and echo pins, and is the simplest mode to use. All code examples for the SRF04 will work for the SRF05 in this mode. To use this mode, just leave the mode pin unconnected - the SRF05 has an internal pull up resistor on this pin.



*Figure 06: Pin of SRF05 sensor*

**Single pin for both Trigger and Echo**

This mode uses a single pin for both Trigger and Echo signals, and is designed to save valuable pins on embedded controllers. To use this mode, connect the mode pin to the 0v Ground pin. The echo signal will appear on the same pin as the trigger signal. The SRF05 will not raise the echo line until 700uS after the end of the trigger signal. You have that long to turn the trigger pin around and make it an input and to have your pulse measuring code ready. The PULSIN command found on many popular controllers does this automatically.



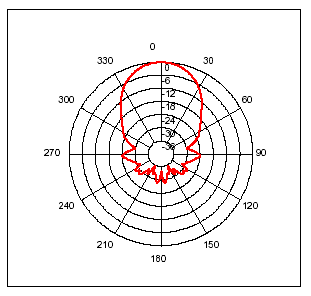
*Figure 07: Timing working of ultrasonic sensors*

The SRF05 Timing diagrams are shown above for each mode. You only need to supply a short 10uS pulse to the trigger input to start the ranging. The SRF05 will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo line high (or trigger line in mode 2). It then listens for an echo, and as soon as it detects one it lowers the echo line again. The echo line is therefore a pulse whose width is proportional to the distance to the object. By timing the pulse it is possible to calculate the range in inches/centimeters or anything else. If nothing is detected then the SRF05 will lower its echo line anyway after about 30mS.

The SRF05 can be triggered as fast as every 50mS, or 20 times each second. You should wait 50ms before the next trigger, even if the SRF05 detects a close object and the echo pulse is shorter. This is to ensure the ultrasonic "beep" has faded away and will not cause a false echo on the next ranging.

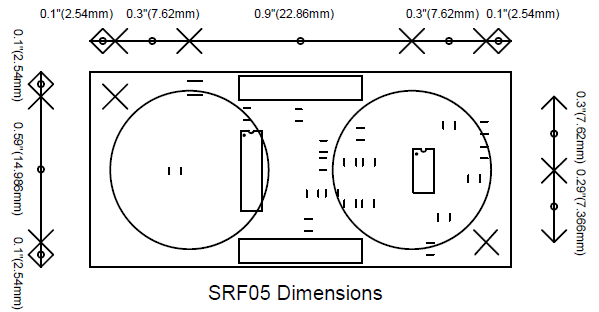
**Changing beam pattern and beam width**

The beam pattern of the SRF05 is conical with the width of the beam being a function of the surface area of the transducers and is fixed.  The beam pattern of the transducers used on the SRF05, taken from the manufacturers’ data sheet, is shown below.

0

*Figure 08: Range of ultrasonic sensor*

**Dimensions**



*Figure 09: Dimensions of ultrasonic sensor*

**Placing Distance Sensors**

In Project, there are 3 ultrasonic sensors in front and 1 ultrasonic sensors at the rear of the car and 2 sensors will place in 2 sides of the car. In the head of the car, 3 ultrasonic sensors will put slanted to 17 degree of each other to calculate the distance from the wheels to the obstacles. Denpending on the distance fo an obstacle to each sensor, we determine how self-driving car move suitable.

In project, we can recognize that when we put the sensor slanted to 17 degree to each other, its test region is enough wide to detect the obstacles when car go ahead.

How self-driving car to behave when facing an obstacles

There are 4 different case when the sensor detect obstacles:

|  |  |  |
| --- | --- | --- |
| **Case 1** |  | All 3 front ultrasonic sensors detect obstacles.  🡪 The car will turn left: θa = (Ox) |
| **Case 2** |  | Only left sensor does not detect obstacles.  🡪 Car will turn left = ( Ox) |
| **Case 3** |  | Only left sensor detects obstacles  🡪 The car will go straight forward |
| **Case 4** |  | Only mid sensor detects obstacles  🡪 Car will turn left 60-degree angle (=- 60) then adjust |

**Angles and Velocity which the car will turn when the car meets an obstacle:**

The car’s velocity is 50 cm/s



Differential velocity between A and B:

A and B will crash at time:

If obstacle VC = VA:



*Figure 10: Angel the car make when moving*

**Powering the Distance Sensor**

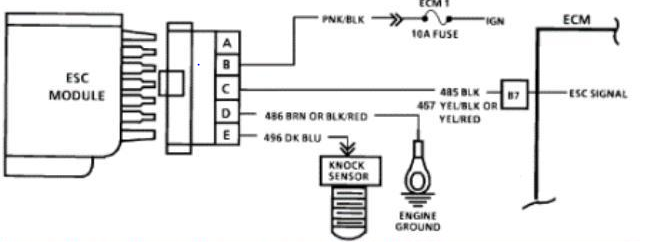
After several different attempts and methods of powering the distance sensor, we decided at last to use Li-po battery 7.4Vol through resistor to power it a voltage 5Vol. Since we found that the distance sensor drives a lot of current (typically 30mA), powering it off of the MCU posed problems to reliable MCU operation. Thus we ended up using another power source in the Li-po battery. However, the distance sensor output needs a common ground in order for the MCU to interpret the data correctly. The grounds of both the MCU and the 5V battery pack were thus connected so that distance sensor could be properly interpreted. This sort of unfavorable connection in theory could cause random and unexpected behavior.

### ESC Circuit

An electronic speed control or ESC is an electronic circuit with the purpose to vary an electric motor's speed, its direction and possibly also to act as a dynamic brake. ESCs are often used on electrically powered radio controlled models, with the variety most often used for brushless motors essentially providing an electronically generated three-phase electric power low voltage source of energy for the motor.

An ESC can be a stand-alone unit which plugs into the receiver's throttle control channel or incorporated into the receiver itself, as is the case in most toy-grade R/C vehicles. Some R/C manufacturers that install proprietary hobby-grade electronics in their entry-level vehicles, vessels or aircraft use onboard electronics that combine the two on a single circuit board.

Regardless of the type used, an ESC interprets control information not as mechanical motion as would be the case of a servo, but rather in a way that varies the switching rate of a network of field effect transistors, or FETs. The rapid switching of the transistors is what causes the motor itself to emit its characteristic high-pitched whine, especially noticeable at lower speeds. It also allows much smoother and more precise variation of motor speed in a far more efficient manner than the mechanical type with a resistive coil and moving arm once in common use.



*Figure 11: Structure of ESC module*

Most modern ESCs incorporate a battery eliminator circuit (or BEC) to regulate voltage for the receiver, removing the need for separate receiver batteries. BECs are usually either linear or switched mode voltage regs in the broader sense are PWM controllers for electric motors. The ESC generally accepts a nominal 50 Hz PWM servo input signal whose pulse width varies from 1ms to 2ms. When supplied with a 1 ms width pulse at 50 Hz, the ESC responds by turning off the DC motor attached to its output. A 1.5 ms pulse-width input signal drives the motor at approximately half-speed. When presented with 2.0 ms input signal, the motor runs at full speed.

**Powering the ESC**

From our manual testing of the RC cars servo motor, we found that a voltage greater than approximately 3.8V was able to turn the actual motor in a direction. Since the voltage across the output pins is determined by the supply voltage to the H-Bridge, we wanted the supply voltage to be in the range of 4-5V for reasonable turning operation. We wanted to avoid introducing a new battery power source and since the MCU power source is out of the question due to the need to isolate the MCU from the servo motor, the car battery was used to power the H-Bridge circuit. However, since the car battery is rated at 7.5V, several diodes were placed in series between the car battery and the H-Bridge supply voltage pin to drop the voltage to an acceptable value. Dropping the voltage from 7.5V was necessary since and excessive voltage applied across the servo motor caused mechanical strain and produced a clicking noise as the car attempts to turn more than it can. In fear that this strain would damage the RC car, we made sure to add the diodes in to reduce the voltage.

### Color Sensor TCS3200

The TCS3200 and TCS3210 programmable color light-to-frequency converters that combine configurable silicon photodiodes and a current-to-frequency converter on a single monolithic CMOS integrated circuit. The output is a square wave (50% duty cycle) with frequency directly proportional to light intensity (irradiance).

The full-scale output frequency can be scaled by one of three preset values via two control input pins. Digital inputs and digital output allow direct interface to a microcontroller or other logic circuitry. Output enable (OE) places the output in the high-impedance state for multiple-unit sharing of a microcontroller input line.

In the TCS3200, the light-to-frequency converter reads an 8 x 8 array of photodiodes. Sixteen photodiodes have blue filters, 16 photodiodes have green filters, 16 photodiodes have red filters, and 16 photodiodes are clear with no filters.

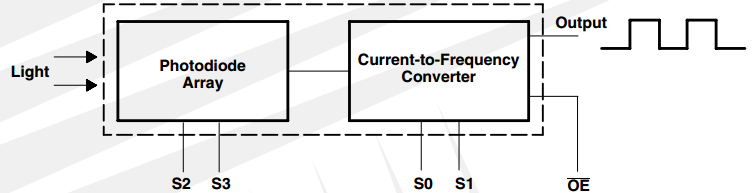
In the TCS3210, the light-to-frequency converter reads a 4 x 6 array of photodiodes. Six photodiodes have blue filters, 6 photodiodes have green filters, 6 photodiodes have red filters, and 6 photodiodes are clear with no filters.

The four types (colors) of photodiodes are interdigitated to minimize the effect of non-uniformity of incident irradiance. All photodiodes of the same color are connected in parallel. Pins S2 and S3 are used to select which group of photodiodes (red, green, blue, clear) are active. Photodiodes are 110 **μ**m x 110 **μ**m in size and are on 134-**μ**m centers.

**Characteristic of TCS3200**

* High-Resolution Conversion of Light
* Intensity to Frequency
* Programmable Color and Full-Scale Output Frequency
* Communicates Directly With a Microcontroller
* Single-Supply Operation (2.7 V to 5.5 V)
* Nonlinearity Error Typically 0.2% at 50 kHz
* Stable 200 ppm/C Temperature Coefficient
* Low-Profile Lead (Pb) Free and RoHS Compliant Surface-Mount Package

**Functional Block Diagram**



*Figure 12: Functional Block Diagram of TCS3200*

**Powering the Color Sensor:**

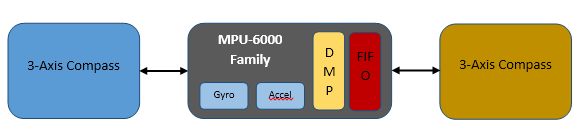
It the same Distance sensor, we decided at last to use Li-po battery 7.4Vol through resistor to power it a voltage 5V.

### MPU-6050 Six-Axis (Gyro + Accelerometer)

The MPU-6050™ parts are the world’s first MotionTracking devices designed for the low power, low cost, and high-performance requirements of smartphones, tablets and wearable sensors.

The MPU-6050 incorporates InvenSense’s MotionFusion™ and run-time calibration firmware that enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discrete devices in motion-enabled products, guaranteeing that sensor fusion algorithms and calibration procedures deliver optimal performance for consumers.

The MPU-6050 devices combine a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die, together with an onboard Digital Motion Processor™ (DMP™), which processes complex 6-axis MotionFusion algorithms. The device can access external magnetometers or other sensors through an auxiliary master I²C bus, allowing the devices to gather a full set of sensor data without intervention from the system processor. The devices are offered in a 4 mm x 4 mm x 0.9 mm QFN package.



*Figure 13: MPU-6000 Family Block Diagram*

The InvenSense MotionApps™ Platform that comes with the MPU-6050 abstracts motion-based complexities, offloads sensor management from the operating system, and provides a structured set of APIs for application development.

For precision tracking of both fast and slow motions, the parts feature a user-programmable gyro full-scale range of ±250, ±500, ±1000, and ±2000 °/sec (dps), and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g. Additional features include an embedded temperature sensor and an on-chip oscillator with ±1% variation over the operating temperature range.

### Power Supply

Due to the low cost and relatively low complexity of our design, this project can easily be rebuilt by those interested in tinkering around with an RC car. The following sections describe the hardware that is used in detail and explains how they are set up.

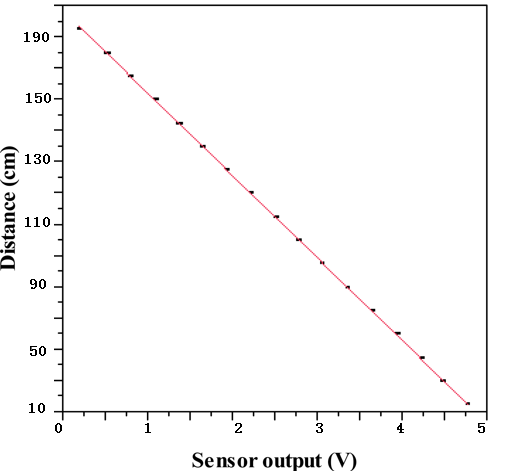
The RC car used in our project was the cheapest RC car we could find at our local Stormracer. It is a 4-wheel car with rear-wheel drive and front-wheel steering. Since a datasheet for the hardware was not provided with it, we had to manually test the cars connections and figure out how to properly operate its motor functions ourselves. Given the nature of our project, the RF functionality of the car was scrapped and the receiver board on the car was removed. Only 8 wires had to be tested to determine their function - 2 wires for the car battery, 3 wires for the rear-wheel DC motor, and 3 wires for the front-wheel servo motor.

The voltage across the car battery hovers around 7.4V, which was expected since the car requires a li-po batteries with 7.4Vol. To test the servo motor wires, we applied different voltages across it starting low and incrementing the voltage until a turning response by the motor was achieved. The servo motor turns in one direction with a positive voltage across it, and turns the other direction with a negative voltage across it, but does not have the capability to turn at varying angles. The servo motor begins responding at around a 3.3V threshold but reaches the max turn angle at a much slower rate. Applying a larger voltage across the motor causes the motor to reach the set turn angle faster. So we use 5Vol apply to servo to achieve the highest performance.

Measurements were taken to determine the effect of varying voltages across the DC motor. Using parts of the tachometer circuit that we built in Lab 4 for this class, we were able to get a relatively accurate measurement of the cars wheel RPM for a given DC voltage across the motor. The data gathered shows us that the threshold voltage for the DC motor to power actual motion of the wheels is around 0.9V and that the voltage-to-RPM relationship is linear.

**Ultrasonic Sensor**

The SRF05 sensor that we used is a very handy, cheap, and easy-to-use piece of hardware that can measure distances in a range from 3 to 200 centimeters. The sensor only has 4 connections that need to be interfaced with power supply, power ground, Trigger and Echo. Given a supply voltage optimally between +4.5 to +5.5 volts, the sensor outputs its measurement of distance as an analog voltage between 0V and 5V as shown in.

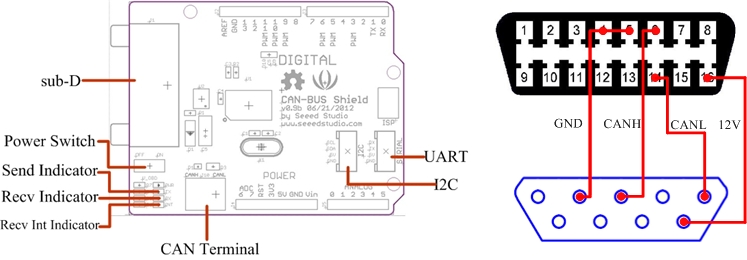


*Figure 14: Analog input of ultrasonic sensor*

**Color sensor**

The TCS3200 sensor that we used is a very handy, cheap, and easy-to-use piece of hardware that can measure distances in a range 2 centimeters. The sensor has 8 connections that need to be interfaced with power supply, power ground, 1 OE, 1 output frequency, 2 output frequency scaling selection inputs, and 2 photodiode type selection inputs. Given a supply voltage optimally between +2.7 to +5.5 volts.

### CAN Module (Controller Area Network)



*Figure 15: Structure of CAN BUS SHIELD module*

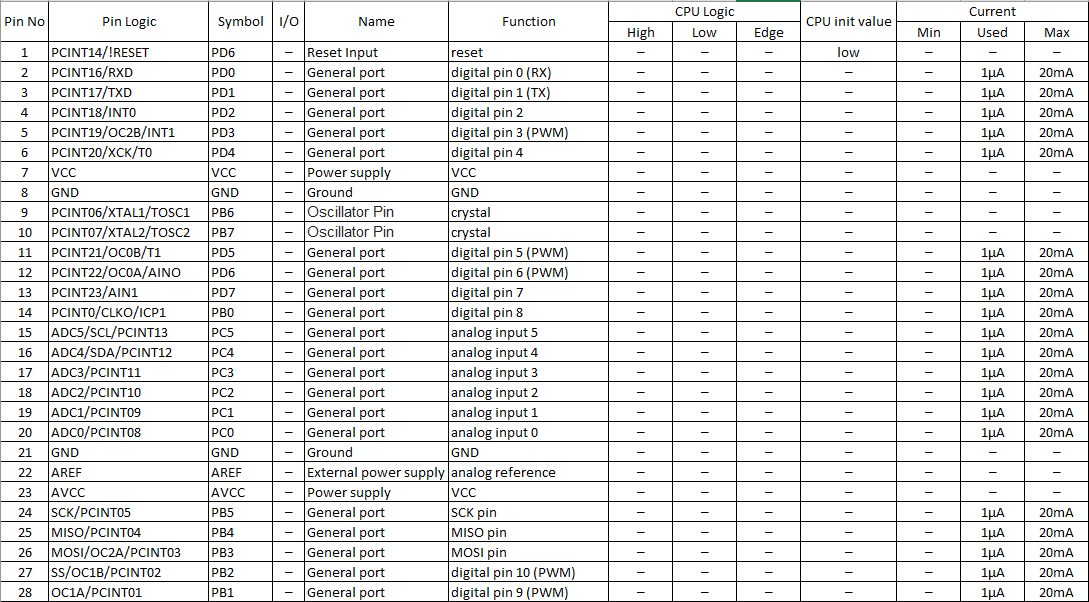
The CAN-BUS Shield provides your Arduino with CAN-BUS capabilities and allows you to hack your vehicle. This shield allows you to poll the ECU for information including coolant temperature, throttle position, vehicle speed, and engine rpms. You can also store this data or output it to a screen to make an in-dash project.

It uses the Microchip MCP2515 CAN controller with the MCP2551 CAN transceiver. CAN connection is via a standard 9-way sub-D for use with OBD-II cable. Ideal for automotive CAN application.

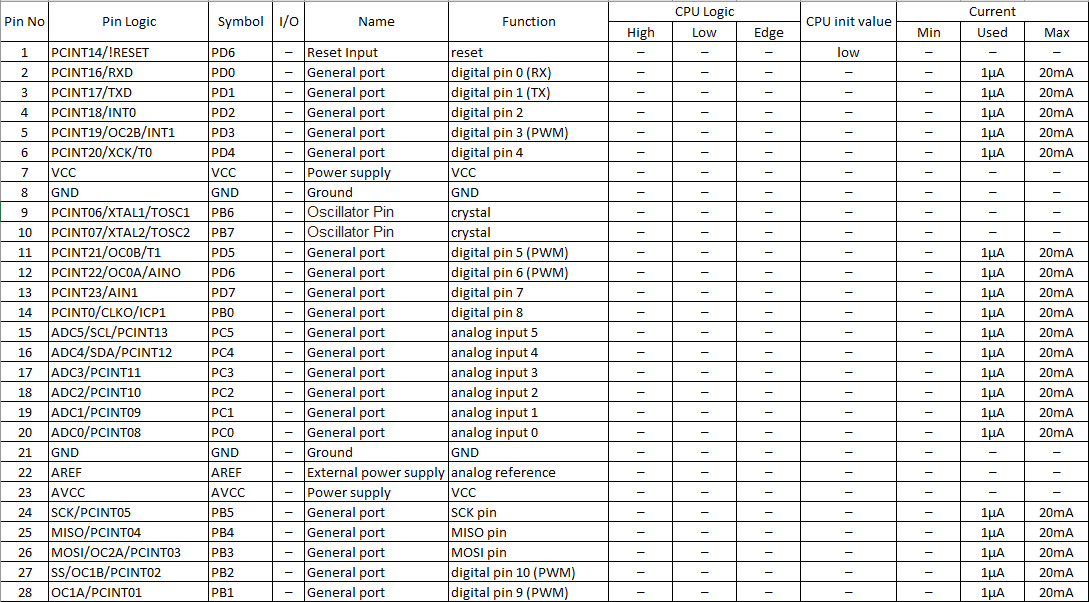
**Features**

* CAN v2.0B up to 1 Mb/s
* High speed SPI Interface (10 MHz)
* Standard and extended data and remote frames
* CAN connection via standard 9-way sub-D connector
* Power can supply to Arduino by sub-D via resettable fuse and reverse polarity protection.
* Socket for EM506 GPS module
* Micro SD card holder
* Connector for serial LCD
* Reset button
* Joystick control menu navigation control
* Two LED indicator

### MCU



*Figure 16: Pin mapping MCU 01:*



*Figure 17: Pin mapping MCU 02:*

## Hardware Issues

* Powering the Distance Sensor: We had many problems with powering the distance sensor which we deem is largely due to the current draw of typically 30mA that is relatively much higher than our other hardware besides the motors. At first we decided to use the MCU’s 5V power lines to power it since the distance sensor operating power supply voltage is between +4.5 - 5.5V. The high current draw caused a loss of power to the MCU which led to the MCU resetting continuously. To remedy this, a large capacitor at around 330uF was placed in parallel with the MCU power lines. This solved the issue with power to the MCU and the distance sensor worked properly. When we began integrating the camera to our system, however, because the camera too is powered off of the MCU power lines we found that the distance sensor was still introducing some noise that made it impossible for the a number of TCS3200 sensor. We then decided to add another battery source to get rid of this noise issue. In order for the distance sensor’s output to be interpreted correctly, however, it needs the MCU reference ground so the negative terminal of our distance sensor’s battery source was connected to MCU ground. In theory, this setup could have presented unexpected errors in our system, but since it worked after testing, we did not delve further into the issue.
* Battery Power Loss: Using batteries while testing our system for extended periods of time proved difficult at times since as the voltage across the battery terminals deteriorated over time, our system behaved differently than expected unless it was recalibrated. In particular losses in power from the sources that power the MCU and the motors caused much frustration since we often misinterpreted it has problems in our hardware design or damage to our circuitry. In the end, however, being aware of this issue and having a voltmeter on hand made this issue less of a problem.
* Road Material: Another issue related to lighting arose when testing our system. Initially we had used tape to layout the road lines that our car was to maneuver between. However, the glare from the tape material caused inaccurate readings by the color sensor and prevented the system from working properly. A material that was dark and had no glare was required for proper system operation. Our cheap solution was to use a dark paper material and tape it down to form our track.

**IV. Software Design**

## Obstacles Avoidance Algorithm



*Figure 18: Algorithm of avoid obstacles method*

## Auto Find Target Algorithm



Figure 19: Square matrix map demo

In square matrix, we choice cross road to a point in dijkstra’s method. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

It is also employed as a subroutine in other algorithms such as Johnson's.

Dijkstra's original algorithm does not use a min-priority queue and runs in time (where is the number of nodes). The idea of this algorithm is also given in (Leyzorek et al. 1957). The implementation based on a min-priority queue implemented by a Fibonacci heap and running in (where is the number of edges).

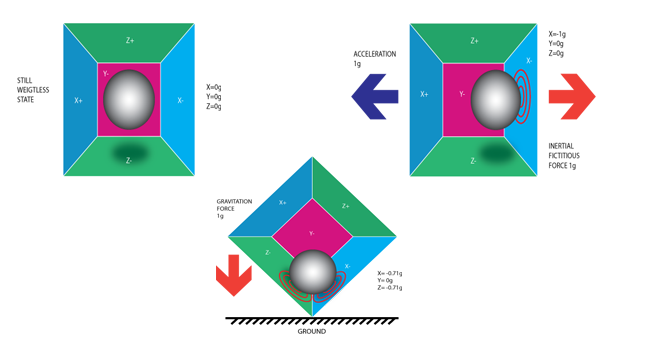
Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance toB (through A) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

## MPU-6050

IMU sensors usually consists of two or more parts. Listing them by priority, they are: accelerometer, gyroscope, magnetometer and altimeter. The MPU 6050 is a 6 DOF (Degrees of Freedom) or a six axis IMU sensor, which means that it gives six values as output. Three values from the accelerometer and three from the gyroscope. The MPU 6050 is a sensor based on MEMS (Micro Electro Mechanical Systems) technology. Both the accelerometer and the gyroscope is embedded inside a single chip. This chip uses I2C (Inter Integrated Circuit) protocol for communication.

How does an accelerometer work?

****

*Figure 20: Trend of MPU6050*

An accelerometer works on the principle of piezo electric effect. Here, imagine a cuboidal box, having a small ball inside it, like in the picture above. The walls of this box are made with piezo electric crystals. Whenever you tilt the box, the ball is forced to move in the direction of the inclination, due to gravity. The wall with which the ball collides, creates tiny piezo electric currents. There are totally, three pairs of opposite walls in a cuboid. Each pair corresponds to an axis in 3D space: X, Y and Z axes. Depending on the current produced from the piezo electric walls, we can determine the direction of inclination and its magnitude. For more information check [this](https://www.filterbypass.me/s.php?k=YQKiltY58vAdq9cjAvXPmA0yfUxpIXzcji4wVVq0ckGoJyWShyM%3D&b=13).

How does a gyroscope work?

****

Figure 21: Piezo Electric Gyroscope

Gyroscopes work on the principle of Coriolis acceleration. Imagine that there is a fork like structure, that is in constant back and forth motion. It is held in place using piezo electric crystals. Whenever, you try to tilt this arrangement, the crystals experience a force in the direction of inclination. This is caused as a result of the inertia of the moving fork. The crystals thus produce a current in consensus with the piezo electric effect, and this current is amplified. The values are then refined by the host microcontroller. Now check this short [video](https://www.filterbypass.me/s.php?k=KBei3M9g8rcPrdIyE%2F7LmAEvd0xpKWPWyVAlC0WvcgWLDxmg0jva&b=13) that explains, how a MEMS gyroscope works.

In this case we use The Euler angles, are three angles introduced by Leonhard Euler to describe the orientation of a rigid body. To describe such an orientation in 3-dimensional Euclidean space three parameters are required. They can be given in several ways, Euler angles being one of them; see charts on SO(3) for others. Euler angles suffer from gimbal lock.

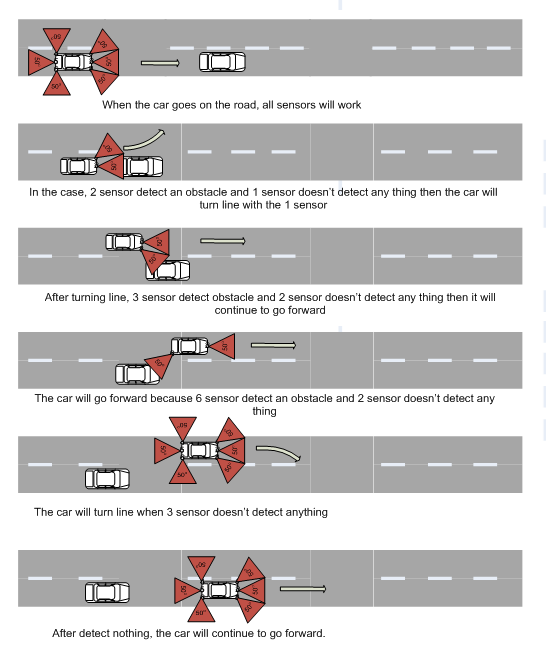
## Ultrasonic Sensor

In Project, there are 3 ultrasonic sensors in front and 1 ultrasonic sensors at the rear of the car and 2 sensors will place in 2 sides of the car. In the head of the car, 3 ultrasonic sensors will put slanted to 17 degree of each other to calculate the distance from the wheels to the obstacles. Denpending on the distance fo an obstacle to each sensor, we determine how self-driving car move suitable.

In project, we can recognize that when we put the sensor slanted to 17 degree to each other, its test region is enough wide to detect the obstacles when car go ahead.

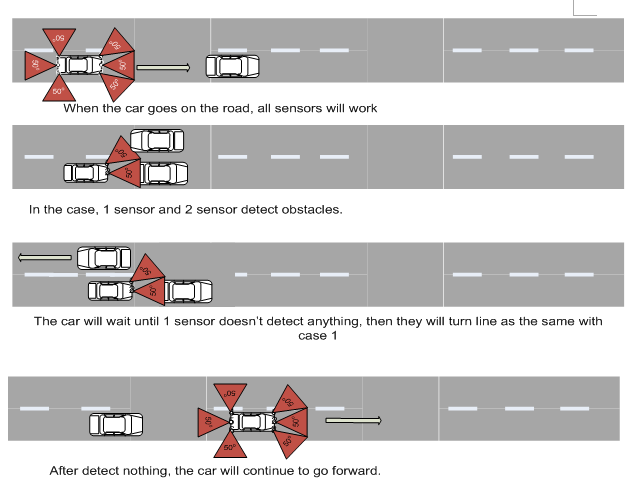
How self-driving car to behave when facing an obstacles

**Case 1:**



*Figure 22: The first case when car avoid obstacles*

**Case 2:**



*Figure 23: The second case when car avoid obstacles*

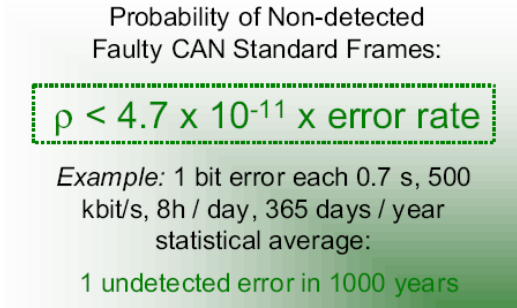
## CAN Controller

Since its inception, the CAN network has been accepted and widespread application in the industrial sector, manufacturing cars, trucks. With time, CAN becomes more common because the efficiency of use, stable, simple, open, and especially low cost. It is used with large data transfers, real-time response and in different environments. Moreover, it has a very stable high-speed transmission.

Therefore they are used in many industries other than the car industry such as agricultural machines, submarines, medical equipment, textile machinery …

Today, CAN has been standardized to ISO11898 standards. Almost every manufacturer big chips such as Intel, NEC, Siemens, Motorola, Maxim IC, Fairchild, Microchip, Philips, Texas Instrument, Mitsubishi, Hitachi, STMicro ... have produced CAN chip, or integrated CAN peripheral into the microcontroller.

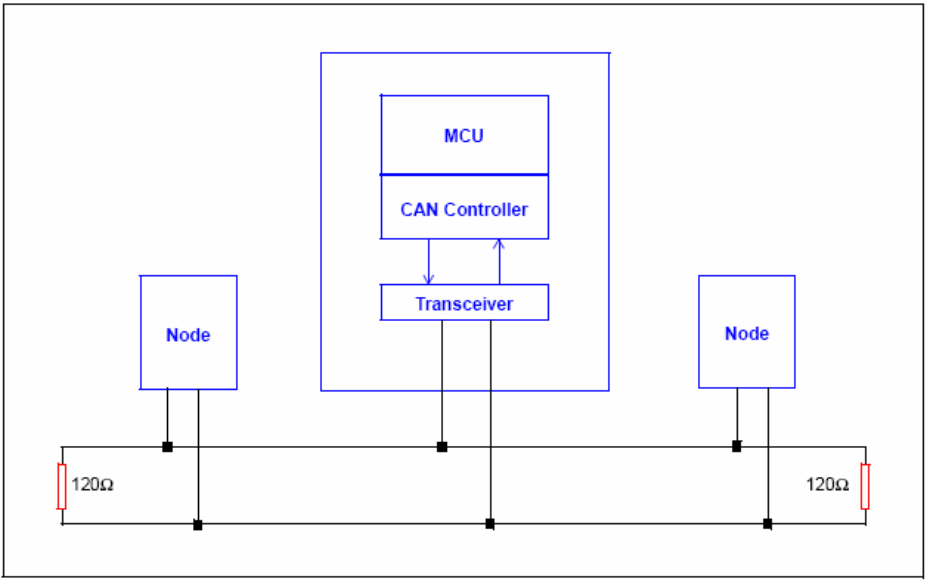
The implementation of the standard CAN become extremely simple thanks to the support from so many chip manufacturers. The most notable point in the CAN standard is the stability and safety (reliability and safety). For having a powerful error detection and handling, CAN error messages almost be detected? According to statistics, probability that a CAN error messages are not detected is:



*Figure 24: Error rate of CAN*

**CAN protocol overview**

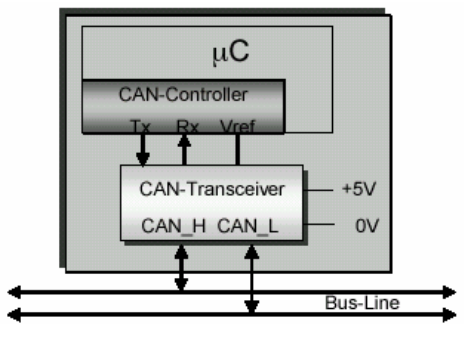
First Standard ISO 11898-2 CAN is to define the nature of the CAN High  
Speed.  
An example of a CAN network in practice



*Figure 25: An example of CAN*

CAN cabling technology has simple transmission lines, minimize the phenomenon of Signal team? The data transfer performed by the pair differential signal lines, which means we measure the difference between the two lines (CAN and CAN L). Bus lines end with resistors 120 ohm (108 ohm and the lowest is a maximum of 132 ohm) at each end CAN network is formed by a group of nodes.

Each node can communicate with any other nodes in the network. The communication is done by the transmitting and receiving data packets – called message. Each type of message in the CAN network is assigned an ID - depending on the priority of that message

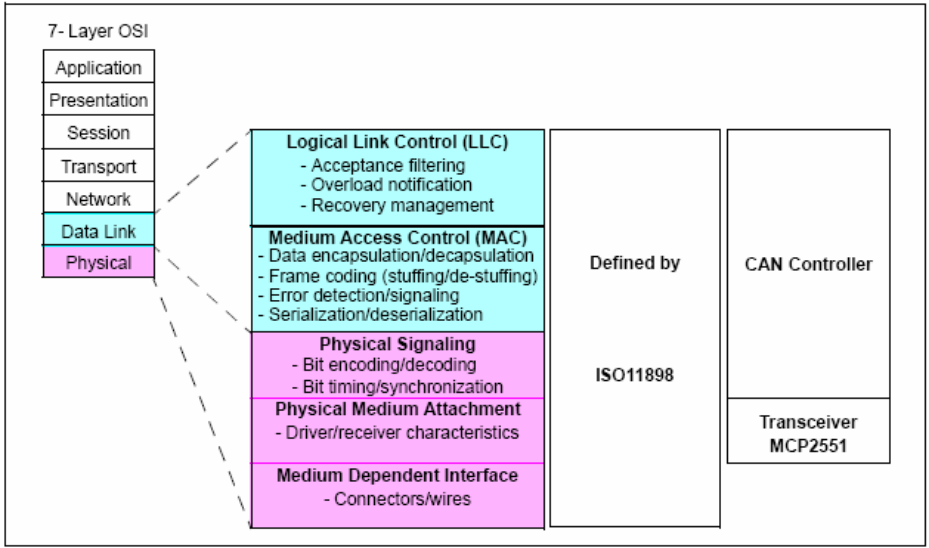


*Figure 26: A CAN node*

CAN is of the type message base system network, different with the base address system, each message type is assigned an ID. In the base system, each node address is assigned to an ID.

Message base system is opener because adding, removing or replacing a node or node group by a more complex node does not affect the whole system. There may be several node receive message and perform a same task. The distributed control system based on the CAN network is openness, easily changed without having to redesign the entire system.

Each node can receive different types of messages, whereas a message can be recognized by many nodes and the job is done in a synchronous distribution system. ID of the message depends on the priority of the message. This allows analysis response time of each message. The importance in the design of embedded systems time.

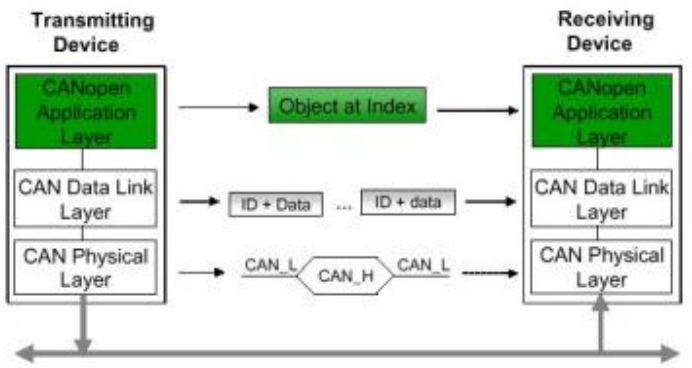


*Figure 27: CAN Model*

ISO11898 standard defines two classes of Physical layer and Data link layer.

Physical layer defines the representation of receiving a bit 0 or bit 1, the timing and co-synchronization.

Data link layer is divided into 2 layers is the logical link control layer of small (LLC) and Medium Access Control (MAC) define transmission frame and arbitration principles to avoid the case that both Master transmitted simultaneously.



*Figure 28: Layers communication.*

In addition, the CAN standard also defines several other mechanisms to check for errors, error handling... test and troubleshooting mechanism is divided into 5 types of errors: Bit error, Stuff error, CRC error, Form error, ACK error.

## Arduino Timers and Interrupts

This tutorial shows the use of timers and interrupts for Arduino boards. As an Arduino programmer you will have used timers and interrupts without detailed knowledge, because all the low level hardware stuff is hidden by the Arduino API. Many Arduino functions use timers, for example the time functions: delay(), millis() and micros() and delayMicroseconds(). The PWM function analog Write () uses timers, as do the tone () and the noTone() functions. And the Servo library uses timers and interrupts.

A timer or to be more precise a timer / counter is a piece of hardware built into the Arduino controller (other controllers have timer hardware, too). It is like a clock, and can be used to measure time events. The timer can be programmed by some special registers. You can configure the prescaler for the timer, or the mode of operation and many other things.

All timers depend on the system clock of your Arduino system. Normally the system clock is 16MHz, but for the Arduino Pro 3.3V it is 8 MHz. So be careful when writing your own timer functions. The timer hardware can be configured with some special timer registers. In the Arduino firmware all timers were configured to a 1 kHz frequency and interrupts are generally enabled.

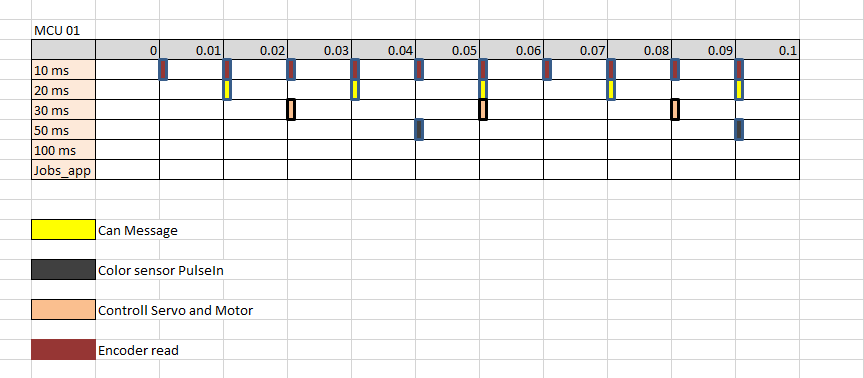
**Timer0:** 8bit timer.

In the Arduino world timer0 is been used for the software Sketch timer functions, like \_\_delay()\_\_,\_\_millis()\_\_ and \_\_micros()\_\_. If you change timer0 registers, this may influence the Arduino timer function. So you should know what you are doing.

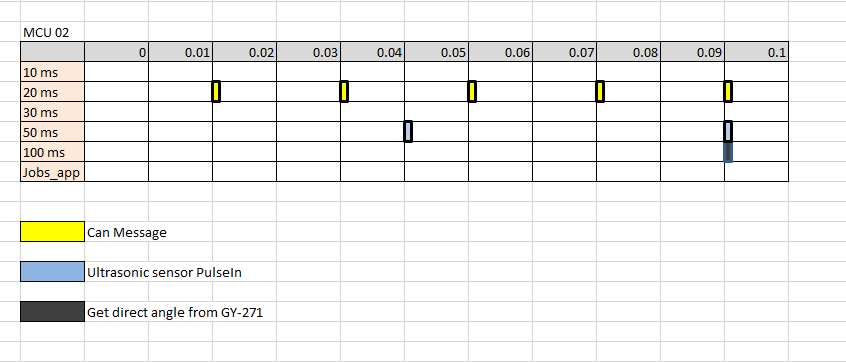
**Timer1**: 16bit timer.

In the Arduino world the \_\_Servo library\_\_ uses timer1 on Arduino Uno (timer5 on Arduino Mega).

**Timer2:** 8bit timer like timer0.



*Figure 29: Timer task on MCU 01*



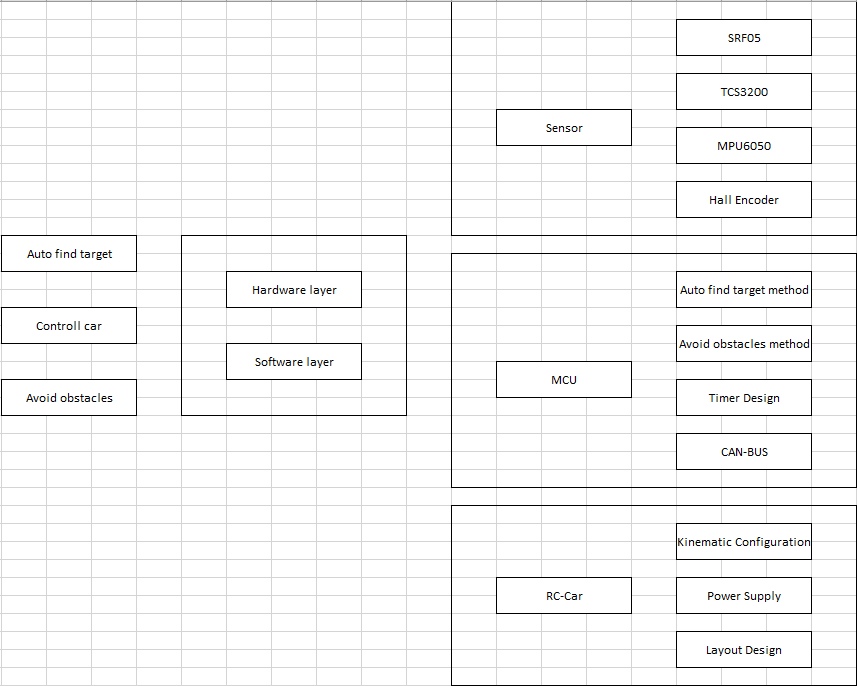
*Figure 30: Timer task on MCU 02*

**V. Evaluation method**

For a car automatically systems running smooth. It should satisfy three requirements:

* Have good auto find target method.
* Can control itself exactly (moving with define velocity, control servo to run straight based on kinematic and dynamics forge on earth).
* Avoid obstacles on its way.

We make a embedded systems can do all jobs above by combine physical layout and software layouts.



Job tasks:

|  |  |
| --- | --- |
| SRF05 Sensor | Ngọc Quang |
| TCS3200 Sensor | Anh Khoa |
| MPU6050 Sensor | Tuấn Anh |
| Hall Encoder | Nguyễn Tín |
| Autofind Target Method | Tuấn Anh |
| Avoid Obstacles Method | Ngọc Quang |
| Timer Design | Tuấn Anh |
| CAN Module | Trọng Nhân |
| Kinematic Configuration | Trọng Nhân |
| Power Supply | Nguyễn Tín |
| Layout Design | Anh Khoa |

**VI. References**

## Datasheets

* [MPU-6050](http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/) Datasheet
* [Atmega 328p datasheet](•%09http:/www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf)
* [SRF05 datasheet](http://www.fgcvme.co.uk/MUSRF05%20Data%20Sheet.pdf)
* [TCS3200 datasheet](https://www.parallax.com/sites/default/files/downloads/28302-TCS3200-doc.pdf)

## Useful Websites

* <http://sophiateam.undrgnd.free.fr/microcontroller/camera/>
* <http://www.seattlerobotics.org/encoder/200011/LineDetect2.htm>
* <https://en.wikipedia.org/wiki/Gimbal_lock>
* <https://en.wikipedia.org/wiki/Euler_angles>