

AUTOMATIC HAND BRAKE SYSTEM

DEPARTMENT OF MECHANICAL ENGINEERING

Abstract :

Hand brake is one of the most important components in vehicles. In general the hand brake is operated manually in our project, We are developing Automatic Hand Brake System for safety purpose. The hand brake engagement and disengagement is done with the help of limit switch ,motor and Gears.

Keywords :

limit switch, motor, microcontroller, Gears, relays, hand brake.

INTRODUCTION :

In cars the hand brake is a latching brake usually used to keep the car stationary. automobiles e-brakes usually consist of a cable directly connected to a brake mechanism on one end and to some type of mechanism that can be actuated by the driver on the other end .the mechanisms is often a hand –operated lever, on the floor on either side of the driver, a pull handle located below and near the steering wheel column, or a pedal located far apart from the other pedals.

Although sometimes known as an emergency brake, using it in any emergency where the footbrake is still operational is likely to badly upset the brake balance of the car and increase the likelihood of loss of control of a vehicle for example by initiating the rear – wheel skid. Additionally, the stopping force provided using the hand brake of or in addition to the footbrake is usually small and would not significantly aid in stopping the vehicle, again because it usually operates on rear wheel while braking .the emergency brake is instead intended for use in case of mechanical failure where the regular footbrake is inoperable or compromised, hopefully with opportunity to apply the brake in a controlled manner to

bring the vehicle to a safe. If gentle half before seeking service assistance The most common use for an automobile emergency brake is to keep the vehicle motionless when it is parked, thus the alternative name, parking brake .car emergency brake have a ratchet locking mechanism that will keep them engaged until a release button is pressed . on vehicles with automatic transmission, this is usually used in concert with parking pawl in the transmission .automotive safety experts recommended the use of both system is required by laws in some jurisdictions .yet many individuals use only the park position on the automatic transmission and not the parking brake.

WORKING PRINCIPLE:

A rack is a toothed bar or rod that can be thought of as a sector gear with an infinitely large radius of curvature. Torque can be converted to linear force by meshing a rack with a pinion; the pinion turns; the rack moves in a straight line. Such a mechanism is used in automobiles to convert the rotation of the steering wheel into the

left-to-right motion of the tie rods. Linear actuation is used to engage the hand brake lever and disengagement is done by spring tension.

AUTOMATIC HAND BRAKE:

The Hand brake lever is coupled with gear setup. A motor used to apply and release the hand brake through gear. Motor driven by the control unit (Micro controller) using sensors. Sensors give a signal to control unit when the vehicles is in rest stage. One limit switch placed near gear lever to find lever in neutral position and another limit switch placed near any one gear to confirm the

wheel is not running state the last is used to detect whether the ignition is ON or OFF. Control unit attach the Hand brake when vehicle is in idle stage that is confirmed by the sensors. Also control unit will release the brake when gear lever is changed from neutral.

MAJOR COMPONENTS :

Gears

Motor

Relay

Arduino

12v power supply

Gears :

A pinion is a type of linear actuator that comprises a pair of gears which convert rotational motion into linear motion. A circular gear called "the pinion" engages teeth on a linear "gear" bar called "the rack"; rotational motion applied to the pinion causes the rack to move, thereby translating the rotational motion of the pinion into the linear motion of the rack. Thus the linear motion is used to actuate the hand brake with the help of small link.

MOTOR:

A dc relies on the fact that like magnet poles attract each other. A coil of wire with a current running through it generates an electromagnetic field aligned with the center of the coil. The motor is used to drive the pinion which drives the rack so that circular motion is converted into linear motion. The source of current is obtained from the 12 v dc battery from the car.

RELAYS:

Relays are the electrically operated switch. Many relays use an electromagnet to mechanically operate a switch,

but other operating principals are also used such as solid state relays. Relays are used where it is necessary to control a circuit by low power signal (with complete electrical isolation between control unit and controlled circuits). We are using the relays for switching the motor ON or OFF. For making automatic hand brake system more efficient, these relays should be operated perfectly in all driving and climatic conditions.

FEASIBILITY STUDY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

- Economic Feasibility
- Technical Feasibility
- Social Feasibility
- Operational Feasibility

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources.

This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity.

The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

OPERATIONAL FEASIBILITY

Operational feasibility is a measure of how well proposed system solves the problems, and takes

advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development. Operational feasibility reviews the willingness of the organization to support the proposed system. This is probably the most difficult of the feasibilities to gauge. In order to determine this feasibility, it is important to understand the management commitment to the proposed project.

Arduino :

The arduino code is actually just plain old c without all the header part (the includes and all). when you press the 'compile' button, the IDE saves the current file as arduino.c in the 'lib/build' directory then it calls a makefile contained in the 'lib' directory.

This makefile copies arduino.c as prog.c into 'lib/tmp' adding 'wiringlite.inc' as the beginning of it. this operation makes the arduino/wiring code into a proper c file (called prog.c).

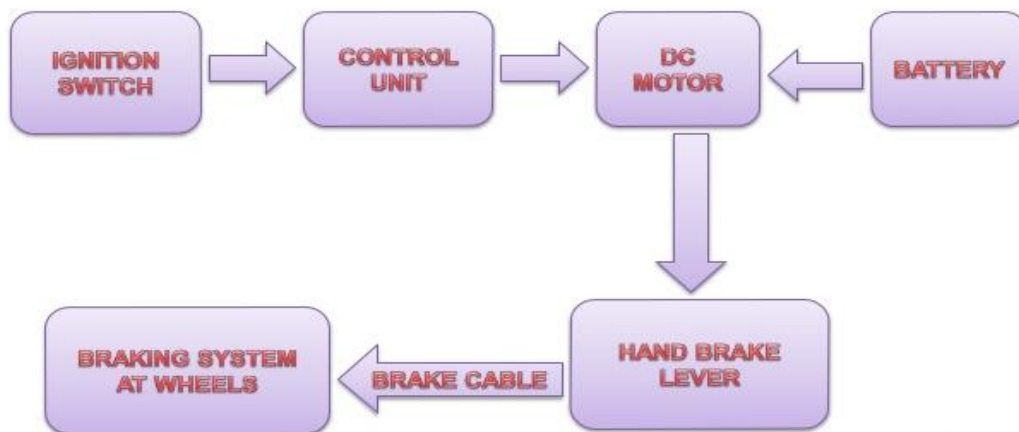
After this, it copies all the files in the 'core' directory into 'lib/tmp'. these files are the implementation of the various arduino/wiring commands adding to these files adds commands to the language

The core files are supported by pascal stang's procyon avr-lib that is contained in the 'lib/avr-lib' directory

At this point the code contained in lib/tmp is ready to be compiled with the c compiler contained in 'tools'. If the make operation is succesfull then you'll have prog.hex ready to be downloaded into the processor.

NOTE:the next release will see each architecture (avr/pic/8051) to treated as a 'plug-in' to the IDE so that the user can just select from a menu the microcontroller board to use and the IDE will pick the right compilation sequence.

BLOCK DIAGRAM :



Objectives:

To design automatic hand brake release mechanism.

To prevent the chances of damage of vehicle parts and to get better efficiency.







To facilitate the release of the brake without any manual effort and during all the necessary conditions inherent in actual movement of vehicle is present.

Our project comprises an electric circuit which consists of control unit, stepper motor and cable. When the vehicle is to be started, the ignition key is locked for ignition, this sends the command to the control unit which actuates the stepper motor and lead screw to release the hand brake based on operating conditions. When the vehicle parked and the key is taken off, this sends a signal to the control unit which reverses the motor direction such that hand brake is applied.

Methodology:

Calculating the load that how much is offered from normal ratchet pawl brake.

Measuring the diameter of brake drum

-  Selecting the suitable design of model.
-  Designing the model as per the dimensions.
-  Selecting the proper electrical circuits and microprocessor.
-  Programming the microprocessor with inherit condition to satisfy.
-  Testing the working model.
-  Correcting the errors.

SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the

process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All

decision branches and internal code flow should be validated. It is the testing of individual software units of the application. it is done after the completion of an

individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure

that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically

aimed at exposing the problems that arise from the combination of components.

FUNCTIONAL TESTING

Functional test can be defined as testing two or more modules together with the intent of finding defects, demonstrating that defects are not present, verifying that the module performs its intended functions as stated in the specification and establishing that a program does what it is supposed to do.

BLACK BOX TESTING

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

WHITE BOX TESTING

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings,

structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

CONCLUSION:

In our project, hand brake is actuated with the help of rack and pinion and proximity sensor based on some conditions. Sensors sense and provides signal to the circuit board which directly drives the motor. The rack and pinion gets activated and lifts the hand and disengages with the help of push lock and spring tension. In future, this could be developing

by adding some of the additional features and also automatic hand brake will be used in all types of automobiles at low cost

IV. PROGRAMMING

setup()

The setup() function is called when a sketch starts. It is used to initialize variables, pin modes, start using libraries, etc. The setup function will only run once, after each powerup or reset of the Arduino board.

Example

```
int buttonPin = 3;
```

```
void setup() {
```

```
  Serial.begin(9600);
```

```
  pinMode(buttonPin, INPUT); }
```

```
void loop() {
```

```
  // ... }
```

LOOP()

After creating a `setup()` function, which initializes and sets the initial values, the `loop()` function does precisely what its name suggests, and loops consecutively, allowing the program to change and respond. It is used to actively control the Arduino board.

Example

```
const int buttonPin = 3;
```

```
// setup initializes serial and the button pin void setup()  
{
```

```
  Serial.begin(9600);
```

```
  return;
```

```
  return value; // both forms are valid
```

Parameters

value: any variable or constant type

Examples:

A function to compare a sensor input to a threshold

```
int checkSensor(){  
  if (analogRead(0) > 400) {
```

```
return 1; else{  
  
return 0;  
  
void loop(){  
// brilliant code idea to test here  
  
return;  
// the rest of a dysfunctional sketch here // this code will  
never be executed
```

#include

#include is used to include outside libraries in your sketch. This gives the programmer access to a large group of standard C libraries (groups of pre-made functions), and also libraries written especially for Arduino.

Note that #include has no semicolon terminator, and the compiler will yield cryptic error messages if you add one.

analogRead()

Description

It reads the value from the specified analog pin. The Arduino board contains a 6 channel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution can be changed using `analogReference()`.

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

Syntax `analogRead(pin)` Parameters

`pin`: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns

`int` (0 to 1023)

Note

If the analog input pin is not connected to anything, the

```
pinMode(buttonPin, INPUT); }
```

```
// loop checks the button pin each time, // and will send  
serial if it is pressed void loop()
```

```
{
```

```
if (digitalRead(buttonPin) == HIGH) Serial.write('H');
```

```
else Serial.write('L');
```

```
delay(1000);
```

```
do - while
```

The do loop works in the same manner as the while loop, with the exception that the condition is tested at the end of the loop, so the do loop will always run at least once.

```
do
```

```
{
```

```
// statement block
```

```
} while (test condition); Example
```

```
do {
```

```
// wait for sensors to stabilize
```

```
delay(50);  
x = readSensors(); // check the sensors  
  
} while (x < 100);
```

Return

It terminates a function and returns a value from a function to the calling function, if desired. Syntax:

value returned by `analogRead()` will fluctuate based on a number of factors (e.g. the values of the other analog inputs, how close your hand is to the board, etc.).

Example

```
int analogPin = 3; // potentiometer wiper (middle  
terminal) connected to analog pin 3  
  
void setup() {  
  
  pinMode(ledPin, OUTPUT); output  
}  
  
void loop() {  
  
  digitalWrite(ledPin, HIGH);  
  delay(1000); // waits for a second digitalWrite(ledPin,
```



```
LOW); // sets the LED off delay(1000); // waits for a  
second
```

```
}
```

Program Used in the System

```
// Include the Stepper Library
```

```
#include<Stepper.h>
```

```
//Setting the speed limits. Upper limit is lim1, Lower  
limit is Lim 2
```

```
int lim1=120,lim2=100;
```

```
float rpmvalue;
```

```
//Define the parameters of stepper motor called stepper1
```

```
Stepper stepper1(200,8,9,10,11);
```

```
int delayvalue=300;
```

```
int stepperspeed=10;
```

```
void setup() {
```

```
Serial.begin(9600); }
```

```
void loop() {
```

```
//define the speed of the stepper  
stepper1.setSpeed(stepperspeed); //read rpm value
```

```
rpmvalue=rpm(20);    Serial.println("Entry    RPM");
Serial.println(rpmvalue);

if(rpmvalue>lim1) {

//Turn the stepper to engage clutch and brake
stepper1.step(200);
do
{

//Lock the motor for some time delay(delayvalue);
rpmvalue=rpm(10);    Serial.println("Delay    Rpm");
Serial.println(rpmvalue);

}

while ( rpmvalue>lim2); //Disengage the clutch and
brake

stepper1.step(-200); }

}

int val = 0;

void setup() {

Serial.begin(9600);
```

```
void loop() {  
  
  // sets the LED on  
  
}  
  
// outside leads to ground and +5V // variable to store the  
value read  
  
val = analogRead(analogPin);  
  
Serial.println(val);  
  
} millis()  
  
int ledPin = 13; 13  
  
// LED connected to digital pin  
  
// setup serial  
  
// read the input pin // debug value
```

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

Returns:

Number of milliseconds since the program started (unsigned long).

Example:

```
unsigned long time;
```

```
void setup(){ Serial.begin(9600);
```

```
}
```

```
void loop(){
```

```
Serial.print("Time: ");
```

```
time = millis();
```

```
//prints time since program started Serial.println(time);
```

```
// wait a second so as not to send massive amounts of
```

```
data delay(1000);
```

```
}
```

```
delay()
```

Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

Syntax:

```
delay(ms)
```

Parameters:

ms: the number of milliseconds to pause (unsigned long)

Example:

```
//Subprogram to calculate the RPM unsigned long int  
rpm(byte i)  
{  
  
byte rpmcount,count=0; unsigned long timeold,time;  
float rpmval;  
int sensorval;  
  
rpmcount = 1;  
  
timeold = millis();  
  
while(rpmcount<=i) {  
  
delay(100); sensorval=analogRead(1); if(count==0)  
  
{  
  
count=1; }  
  
}  
  
if(sensorval<800) {
```

```
count=0; }
```

```
time=millis()-timeold;
```

```
rpmval = (60000/time)*rpmcount;
```

```
return rpmval;
```

```
const int ledPin = 13; //LED connected to digital pin 13
```

```
const int Sensor = A0; // Sensor connected to analog pin
```

```
A0 const int threshold = 100; // Threshold is set to 100
```

```
int sensorReading = 0; // variable to store the value read  
from the sensor pin
```

```
int ledState = LOW; // variable used to store the last LED  
status, to toggle the light
```

```
void setup()
```

```
{
```

```
pinMode(ledPin, OUTPUT); // declare the ledPin as  
OUTPUT
```

```
}
```

```
void loop()
```

```
{  
  
// read the sensor and store it in the variable  
sensorReading:  
  
sensorReading = analogRead(Sensor);  
  
// if the sensor reading is greater than the threshold:  
  
if (sensorReading >= threshold)  
  
{  
  
// toggle the status of the  
  
ledPin: ledState = !ledState;  
  
// update the LED pin :  
  
digitalWrite(ledPin, ledState);  
  
delay(10000); // delay  
  
}  
  
else  
  
{
```

digitalWrite (ledPin, ledState); // the initial state of LED
i.e. LOW.

}

}

int IRSensor = 2; // connect ir sensor to arduino pin 2

int LED = 13; // conect Led to arduino pin 13void setup()

{

pinMode (IRSensor, INPUT); // sensor pin INPUT

pinMode (LED, OUTPUT); // Led pin

OUTPUT}void loop()

{

int statusSensor = digitalRead (IRSensor);

if (statusSensor == 1)

digitalWrite(LED, LOW); // LED LOW

}

Else

{

digitalWrite(LED, HIGH); // LED High

}

}

int photoDiode=2;

int GreenLed=13;

int senRead=0;

int SenseRate=905;

void setup()

{

pinMode(photoDiode,OUTPUT);

pinMode(GreenLed,OUTPUT);

pinMode(12,OUTPUT);

digitalWrite(photoDiode,HIGH);

```
digitalWrite(GreenLed,LOW);
```

```
Serial.begin(9600);
```

```
}
```

```
void loop()
```

```
{
```

```
int val=analogRead(senRead);
```

```
Serial.println(val);
```

```
if(val <= SenseRate)
```

```
{
```

```
digitalWrite(12,HIGH);
```

```
digitalWrite(GreenLed,LOW);
```

```
delay(20);
```

```
}
```

```
else if(val > SenseRate)
```

```
{
```

```
digitalWrite(12,LOW);
```

```
digitalWrite(GreenLed,HIGH);
```

```
delay(20);
```

```
}
```