# Autonomous Car Parking

Author: Gabriel Almeida

[gabriel.c94@gmail.com]

Czech Technical University in Prague, Department of Measurement

March 27th 2017

## Introduction:

With the increasing number of cars in our streets, the amount of crashes and accidents caused by human error only grows. This brings the necessity to automate not only driving, but parking also, most drivers have difficulty with parallel parking, which is the type of parking that this project proposes to solve.

## The prototype:

For the development of this project, a remote controlled car has been modified to serve as a prototype, it consists of a frame with 4 wheels, 1 dc motor, and a servo. The dc motor drives the two back wheels, and the servo is used to steer the car front wheels. Two inductive proximity sensors and three ultrasonic sensors were also added to the car frame, the inductive proximity sensors are used to measure the wheel's speed and the ultrasonic sensor to detect surrounding objects.

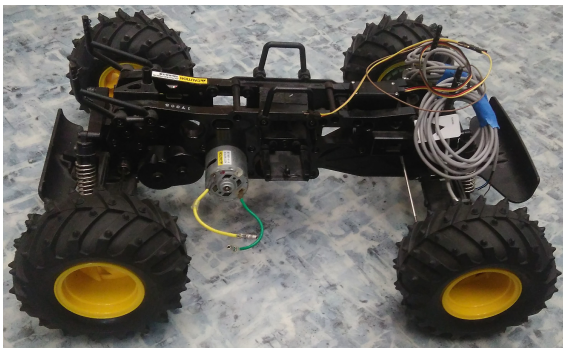

*Illustration 1: The remote control car used*

**Characteristics:**

- Car length: 0.42m
- Front axis: 0.35m
- Back axis: 0.31m
- Wheel circumference: 0.40m

## Desired behavior:

The parking of the car consists proposed in this project consists of two parts, finding and measuring the possible parking spot and parking the car in the spot found. To find the parking spot, the car is programmed the go forward in a straight line, using its side sensor to detect if there are cars present there or not, when there is no car present it starts to count for how long there will be no more cars, and with the speed measured it calculates how big the parking spot is. If the parking spot is big enough, it starts the parking algorithm, if not, it continues going forward looking for another parking spot.

## Hardware:

- DC motor
- Servo
- 3 Ultrasonic sensors
- 2 Inductive proximity sensors
- Breadboard
- H-bridge
- Arduino Mega ADK

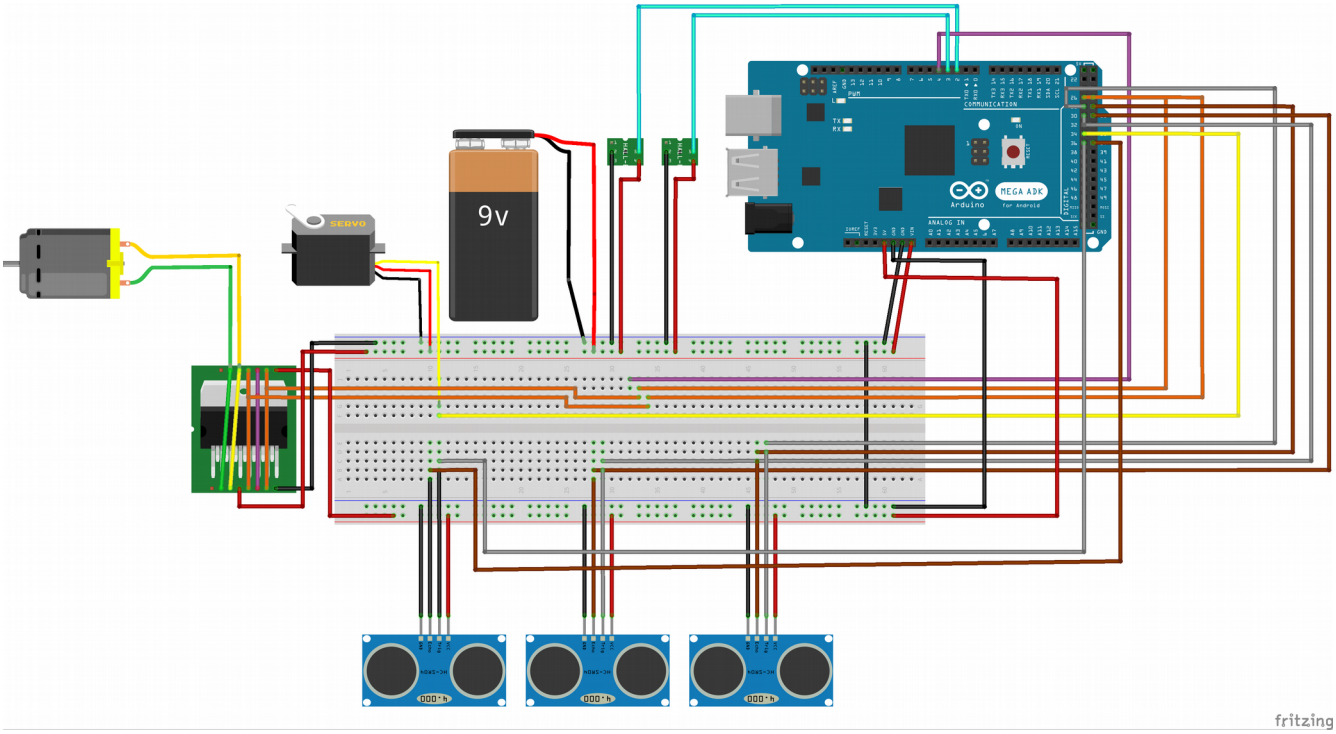


*Illustration 2: Project's full schematics*

## Ultrasonic sensors:

These sensors works by sending an ultrasonic wave and hearing for its echo, the distance is calculated from the time delta from when the wave is sent to when its echo is heard, it was used the NewPing library to use this sensor more easily to measure the object's distance.

*Illustration 3: Ultrasonic sensor*

## Inductive proximity sensors:

These sensors works by creating an electromagnetic field in front of it, if any metal is in a short distance in front of it, it will disturb the electromagnetic field and thus be detected, this sensor is used

together with a metal disc with 20 holes, attached to each one of the wheels, so whenever the car wheel spins, the gaps in the disc will pass in front of the sensor, which makes it possible to tell how many 1/20 of turns the wheel has made, and this information is used to calculate the car speed.

*Illustration 4: Inductive proximity sensor*

## H-bridge:

The h-bridge is used deliver power and to control the rotation direction of the motor, whether it goes forward or backwards, the one used in this project was the L298N, the h-bridge logic power is supplied by the Arduino, while the power passed to the motor comes from the main battery itself. This bridge can drive two motors at 2A each, but as our motor might draw more current than that, the bridge was connected in a parallel configuration, so it can drive one motor at 4A, to do this, it is needed to short the following pair of pins: IN1 and IN4, IN2 and IN3, ENA and ENB, OUT1 and OUT4, OUT2 and OUT3, as shown in the project schematics.

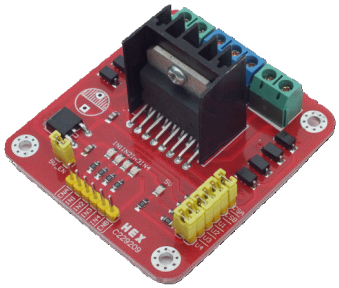

*Illustration 5: H-bridge L298N*

## Servo:

The servo is used to control the car steering, its usage is very straight forward, it is supplied by the main battery directly, and one of its pins connects to the Arduino, which sends the desired angle to the servo. To do this the Servo library was used, to more easily control it.

*Illustration 6: Servo*

## The driving straight problem:

One of the main problems in this project was making the car drive in a straight line, it is the same problem when driving a normal car, you see with your eyes that even though your are holding the steering wheel in a fixed position, the car might be going a little to the left or to the right, due to many reasons. In this small prototype, the problem is even worse, because the wheels are hard to keep aligned, any weight added to any side of the car deeply affects it, and the servo position control is not as precise as needed. Even faced with all this problems, a solution was implemented to deal with it, the rotation of each one of the wheels is measured individually, if one wheel is spinning faster than the other, the car slightly corrects the servo angle, using a proportional control. The implemented solution doesn't work well, because when the car is on very wide turns, the wheel rotation difference isn't detected, because the metal disc only have 20 holes in it.

For example, lets take a 4 meters radius turn, which already affects the straight trajectory too much even in small distances, given that the car roughly 0.3 meters wide, the inner wheel will make a 3.7 m turn. To keep the car on track, we need to correct its path constantly, in this project, the rotation difference is measured every 0.5 seconds. So, the car is making a 4m turn, which has a circumference of $8\pi$ which is around 25.12m and the inner circumference of 23.23m, if we try to correct the car after a portion of the turn, lets say 5°, the outer wheel would have turned 0.34m, and the inner wheel 0.32m, but the turn detection precision is a little over 0.02m since the wheel circumference is a little bigger than 0.4m and the disc have 20 holes in it, so the sensors would tell that their rotations were the same, and the algorithm wouldn't try to correct the car trajectory, and that happens when we are trying to correct its trajectory after it already went 5° of the curve, if we try to correct this earlier the difference is even smaller and won't be noticed also, and if we do this after more than that, the car would be too far from its trajectory already, and the result would be a terrible zigzag.

To be able to solve this problem, more accurate sensors are needed, with this same approach, a disc with more holes would be needed along with a sensor that could detect those very tiny holes in it.

## Parking space problem:

Due to the size and width of the car, it is too wide for its length comparing to normal vehicles, and the fact that the wheels can only turn in short angles, about 30° in each direction, it takes a lot of space to parallel park it, in the experiments made, it was measured that it takes around one meter to be able to park this car comfortably, which is more than twice its size, all the experiments were done using a 0.94m parking spot.

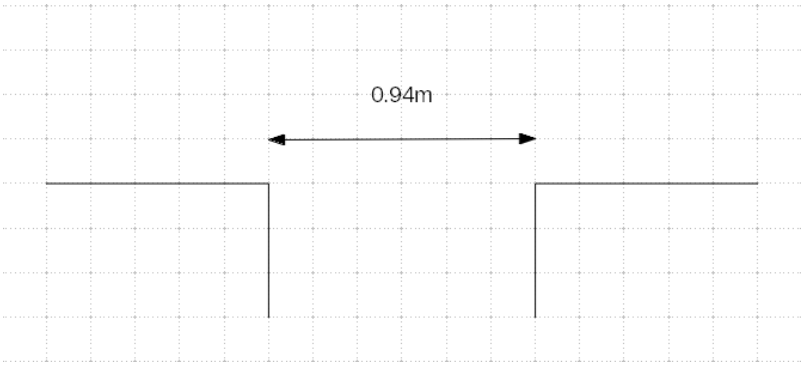


*Illustration 7: Parking spot diagram*

## Parking space measurement:

An important part of the problem was to measure the parking space precisely, because it would be a huge problem if a small parking space is measured incorrectly and the car keeps trying to park somewhere too tight for it, to acquire the measurements, some code were implemented to write to the Arduino EEPROM the value calculated, so then later the values could be read and analyzed on a computer. The calculated standard deviation from those values was 0.055m, which was found to be acceptable.

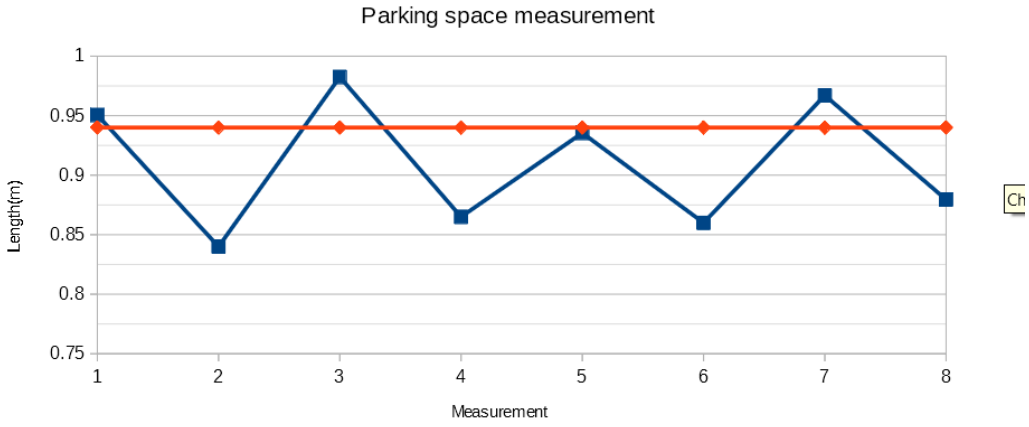


*Illustration 8: Measured values plotted against actual value*

## Power consumption:

When working on this project, it was noted that the battery pack used would drain out quickly, checking the consumption of each component it was clear that this fast drain wasn't normal. One of the possible cause might be the H-bridge, when on, the bridge gets hot, to the point the is not possible to hold some parts of it, the connections were checked but no issue were found.

Before any more work is done to this project, this problem should be checked and solved, because testing and working with a battery that doesn't last long is problematic.

## Algorithm:

The implemented code is broken down as much as possible into small functions, each with a unique purpose, all the code is commented for easier understanding, the main program is divided into two parts, detect the parking spot and park the car.

**Detecting the parking spot:**

The whole program is started by the press of a button on the breadboard, when the program is started it calls the function to find the parking spot( findAndPark() ), it makes the car go forward checking with the side ultrasonic sensor that there is no free space on its side, as soon as it detect the there is no obstacles in its side, it records the time and keeps going forward, until another obstacle is found, then it takes the difference from the time in that moment with the previous recorded time, with this time delta and the speed measured by the sensors it calculates the size of the parking spot, if it is large enough it will start the parking function, if not, it will start this same function again.

**Parking the car:**

When a large enough parking spot is found, the car parking function( sensoredParalelPark() ) is called, this function steers the wheel all the way to the right and makes the car go backwards until no obstacle in detected by the side sensor, then it steers the wheel all the way to the left and continues going backwards, until the back sensors detects the car in the back is closer than 12cm from it, then it turns the wheel again all the way to the right to adjust the car going forward, after the car is adjusted, it goes forward until the front sensor detect that the front car is less than 20cm from it, then the parking is finished.

**Measuring speed:**

The speed measurement in the car is made by the inductive proximity sensors, every time the wheel spins 1/20 of a turn, one of the little metal parts pass through the sensor, which is connected to an interrupt pin in the Arduino, this triggers an interruption routine that increments a counter by one. On top of that, there is an interruption routine programmed to be called every 0.5 seconds, this routines checks how many one twentieth of spin has been since the last time it was checked, then it calculates

the speed based on that, resetting the spin counter to zero for the next call. The routine called every 0.5 seconds is the same routine used to check the difference between the wheels rotations to correct the car trajectory if needed.

## Final considerations:

The autonomous parking is a real world problem that needs a solution, in this project it was noted that it is hard to solve but it is possible. In a few months it was possible to develop the hardware and software to do it in small scale, and the knowledge gathered from it could be used to implement it on real cars. It is a technology already in production in some companies, but it is far from being perfect and fail proof. There is a lot of room for improvement.

## Proposed improvements:

The single best improvement that could be made in this project would be to use a triple axis magnetometer along with the other sensor, this way the direction of the car could be measured more precisely and be corrected faster, solving the driving straight problem.

There were various problems with the inductive proximity sensors used, the main being the fact that whenever there is too much load on the motor, it would draw more current and disturb the functioning of those sensors, making them give false readings or even stop working. Decoupling this sensors from the rest of the circuit or using other types of sensors to measure the speed would be good approaches.

The ultrasonic sensors used are bad for detecting distances in objects not flat shaped, due to the fact that the wave emitter and the receptor are a few centimeters apart from each other, whenever they were facing an object edge it would give odd values for that object distance, often not being able to read the distance at all, this created the need to hard code a lot of workarounds so it would work in the environment set for the tests. Replacing these sensors for more precise infrared distance sensors can yield better results on the object detection, improving the general functioning of the project and simplifying the code.

## For more information:

This project was developed during the UNIGOU program, more information about this program can be found here.

The source code, schematics and other documents are all available in an open repository here, also feel free to contact if interested in this project.