

EE579 Group Project

Advanced Microcontrollers

Christopher Morrison

Adeeb Umar

Stanislas Bertrand

Xavier Galzin

Contents

[Project Specification 2](#_Toc356983580)

[Project Design 3](#_Toc356983581)

[Drive & Steering 3](#_Toc356983582)

[Direction & Distance Feedback 4](#_Toc356983583)

[Input of Values 5](#_Toc356983584)

[Other Design Considerations 5](#_Toc356983585)

[Conclusions 8](#_Toc356983586)

[Hardware 8](#_Toc356983587)

[Steering System 8](#_Toc356983588)

[Forward/Backwards System 9](#_Toc356983589)

[Power System 10](#_Toc356983590)

[Location Sensor 11](#_Toc356983591)

[Serial Communication System 12](#_Toc356983592)

[Software Description 12](#_Toc356983593)

[Input to the car 12](#_Toc356983594)

[Car Movements 14](#_Toc356983595)

[Mouse Measurements 14](#_Toc356983596)

[Math Model of the System 15](#_Toc356983597)

[Path Control 16](#_Toc356983598)

[Bibliography 18](#_Toc356983599)

# Project Specification

The project involved the creation of a refitted Remote Controlled Toy Car, shown in Figure 1. The original car was controlled using a remote control device, containing an IR transmitter. The Toy Car used an IR receiver to receive signals from the remote control, allowing a user to control the car wirelessly from a distance.



Figure 1 - Remote Controlled Toy Car

The car was to be modified in such a way that it can navigate a course, passing an obstacle and finishing within a set distance of a finishing marker, all without intervention from an operator. An example course is shown in Figure 2. As part of completing the course, the distance *d* and the angle θ were to be revealed two minutes before the start time. After these details have been given to the refitted car, it should be able to navigate the course automatically, arriving within a set distance of the final marker. Additionally, the car should be able to complete the task within a specific time.

The refitted car should use an MSP430 G2553 microcontroller to control the car, including the drive, steering and any other associated computation that would need to be completed in order for the car to complete the course – for example location sensing.

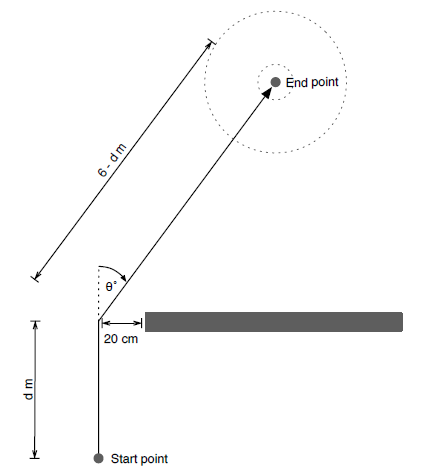


Figure 2 - Example Route, d and θ represent the distance and angle given, respectively.

# 

# Project Design

In order to adhere to the Specification for the project outlined above, the refitted car would have to meet several criteria, namely; to be able to automatically move and turn, to be able to reliably establish its current location and any associated information, and to accept the course distance and angle values as an input. These problems would be overcome using a combination of both hardware and software, taking into consideration any additional problems encountered throughout the refit process.

## Drive & Steering

The movement of the car is controlled by the back wheels providing forward and backward movement, and the front wheels, which turn left or right and allow the car to turn in that direction. In the un-modified car, a DC motor is connected via a gear system to the back wheels, controlling the forward and back movement (circled Red in Figure 3), while two electromagnetic coils control the left and right movements of the steering rack (which has a regular magnet attached) connected to the front wheels (circled Blue in Figure 3). The circuit board seen attached to the top of the body of the car in Figure 3 contains a circuit that relays signals received from the IR receiver through an H-bridge to allow the DC motor to be controlled backwards and forwards. Additionally, two transistors allow the Electromagnet to steer both left and right. By taking advantage of the circuit already attached to the car and its electrical components, space and time could be saved.

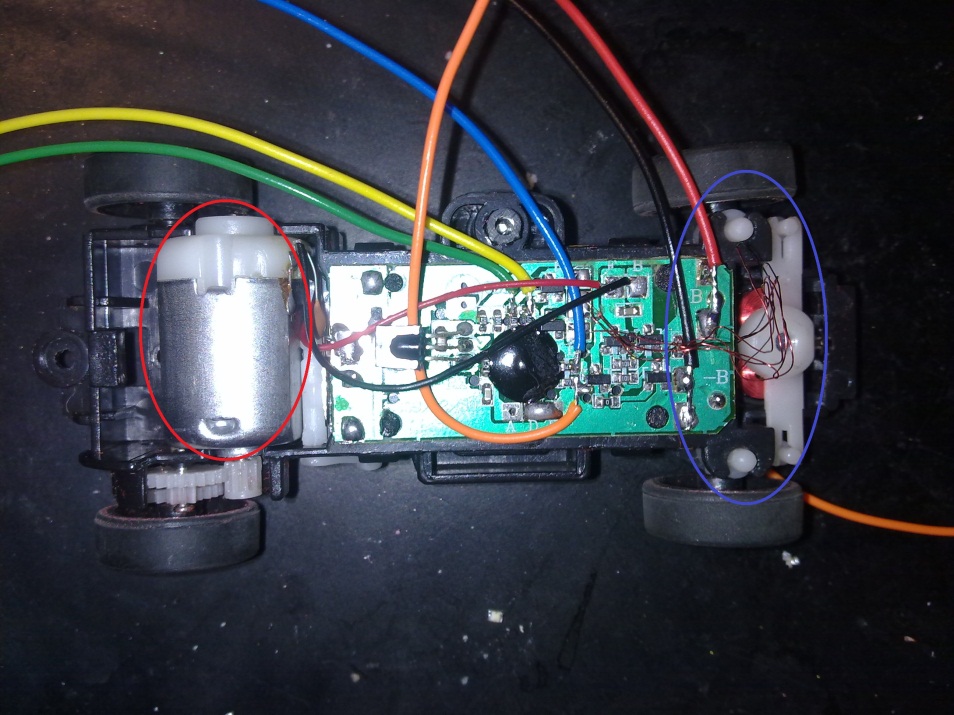


Figure 3 - The Toy Car with the outer shell removed, and additional wires attached.

The refit will also be able to potentially improve the movement of the car. For example, before the refit, the car could only go forward and back at one speed. By adding a power moderation technique, a more granular speed control can be achieved. Another important factor that must be taken into account when refitting the toy car is power. The electromagnets controlling the steering rack use quite a large amount of power when in use, so additional power sources will be needed. The solutions for each of these problems must be evaluated and the best one implemented.

To achieve a more granular speed control, the simplest and best solution is to implement Pulse Width Modulation (PWM) in software. This is done by having a ‘speed control’ function, that takes an integer value (between 0 and 50) as a parameter and controls the mark and space of the outputted PWM signal controlling the car’s forward and back movement, where 50 relates to the maximum recommended speed of the car. This allows the car to speed up and slow down gradually, as well as potentially have a tighter turning circle. As previously mentioned, the un-modified car used an h-bridge to allow bi-directional current flow to the DC motor. By connecting two pins from the MSP430 to each side of the h-bridge already attached to the car, forward and back movement could be achieved using simple output signals from the two microcontroller pins.

## Direction & Distance Feedback

One of the main issues associated with controlling the car is the method by which it can determine its current location, and the method by which it controls its direction and speed of travel appropriately. There are many different ways by which location information could be determined, including using the ‘speed, time and distance’ equation to estimate the distance travelled in a specific time, attempting to navigate towards a ‘beacon’ placed at the end location, or measuring the wheel turns to establish distance travelled.

The main problem with these solutions is the lack of a feedback loop. This can cause the estimated location to vary wildly from the actual location in situations where the reality has not aligned with the theory. For example, the car could be travelling in the wrong direction, or the wheels could be turning on a slippery surface without the car moving. For these reasons it is important to receive feedback from the environment to gain an insight into exactly where the car is located, rather than just estimating.

The best solution to the problem was to connect an optical mouse sensor to the car and use the location feedback that it provided to construct a mathematical model of where the car was in relation to its final location and then calculating the necessary adjustments to be made to the cars speed and direction of travel. As the MSP430 had built-in support for SPI, a mouse that supported that communication protocol was needed. Because the optical sensor had to remain at a specific location above the ground in order to function correctly, and because the original mouse casing was designed to move across a surface, it was decided to attach the optical sensor and the mouse casing in a ‘trailer’ format, pulling the sensor behind the refitted car. This can be seen in Figure 6.

The mathematical model used to plot the course for the car to take uses trigonometry to convert between the various types of co-ordinate systems in use in the system, such as the rectangular form co-ordinates that the optical mouse sensor provides and the polar form angle and distance given at the start of the exercise. The model establishes the location of the final point and calculates the speed and trajectory that the car must travel at to reach the final location.

## Input of Values

As part of the specification, the system must be able to plot its course based only on two input values - a distance *d* and an angle *a*. Since only two minutes were allowed between receiving the values and running the course, a method for entering the values into the system would have to be conceived.

Attaching an input device, such as a keyboard or number pad was a possible option, but due to weight and space considerations was not used. As the MSP430 used for the project has built in serial connectivity, this was decided to be the best way to transmit the values. A text based User Interface would be transmitted over serial to a connected computer, where the values could be inputted at the appropriate time.

An advantage of using a serial communication protocol is that an RF transceiver can be easily added to the serial connections of the MSP430 chip and computer’s USB port, giving the car wireless connectivity to the computer. This stops any problem that might occur with a cable connected to the car getting caught as the car was moving. Additionally, live commands can be sent from the computer to control the car using the computer keyboard as a remote control, although this functionality will be disabled to remain in line with the specification.

## Other Design Considerations

When dealing with such a small physical area on which to attach/install power, control and feedback systems, space considerations can become a major issue. This also affects the balance and speed of the car. For example, putting too much weight on top of the car’s shell would cause it to tip over if it turned with too much speed. For this reason, the large optical sensor remains inside its original plastic shell and is pulled behind the toy car as a trailer. Aside from adjusting the mathematical model plotting the route (as the sensor is three inches behind the car) the performance is not affected.

Another issue inherent to working with microcontrollers is power considerations. Driving an electromagnet, SPI interface, DC motor and microcontroller requires a powerful but stable power supply. The refitted car uses three AA (1.5V) rated batteries to power the steering and DC motor that control the car, and three AAA (1.5V) rated batteries as the power source for the microcontroller, optical mouse sensor, serial communications and the LEDs and buzzer attached to the car.

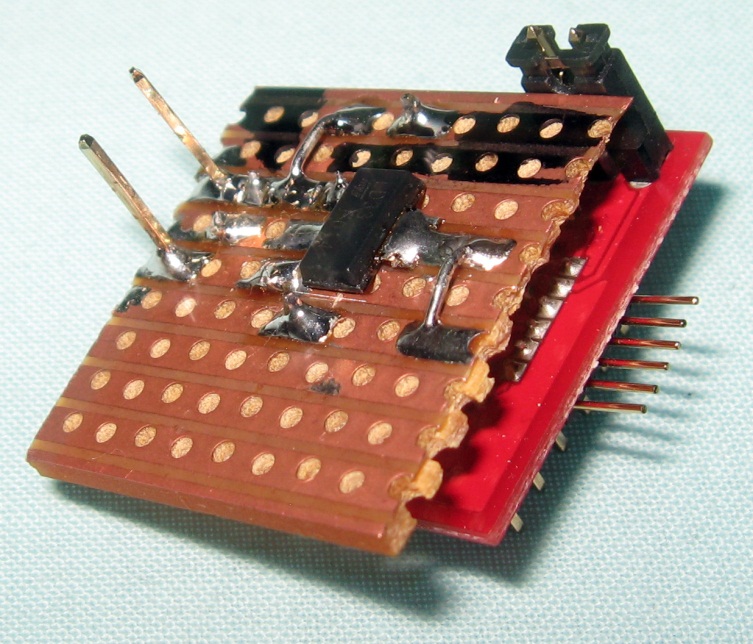
****

Figure 4 - Voltage Regulator

The system supplying power to the microcontroller and optical sensor must supply a consistent power output regardless of battery capacity, rating or current draw throughout the circuit, for this reason a voltage regulator was used, shown in Figure 4. The voltage regulator ensures that things like the PWM and SPI clocks remain at a consistent rate regardless of the charge state of the batteries. The circuit diagram of the voltage regulator connected to the microcontrollers power supply is shown in Figure 5, alongside the power supply used to power the motor and steering.

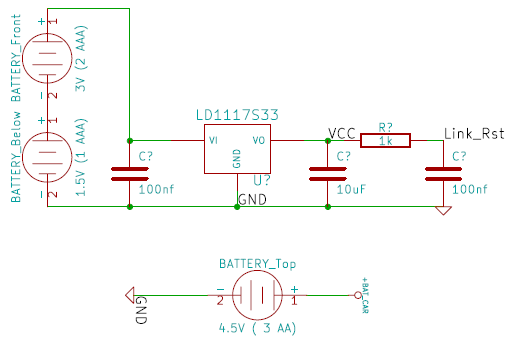


Figure 5 - The power supply, showing the AAA batteries and power regulator (top) and the three AA batteries, (bottom).

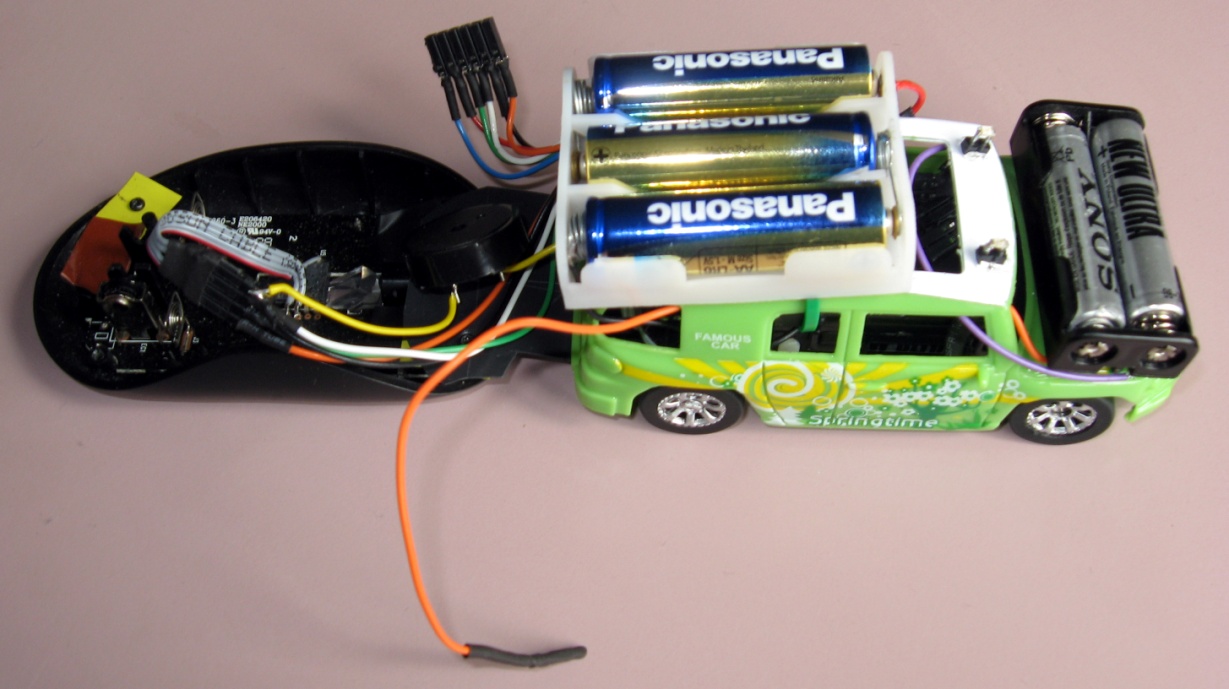


Figure 6 - The final refitted Toy Car with mouse sensor trailer

## Conclusions

By taking advantage of circuitry already attached to the car’s body and wheels, the limited space inside the car could be maximised, as well as saving time in the design and implementation of an equivalent custom-built circuit. The use of a serial based RF transceiver allowed the car to be controlled and ‘programed’ quickly and easily through a serial port opened on the controlling computer. The optical mouse sensor used for the location sensing provided a high resolution location sensor that is designed for moving around on different surfaces and is perfectly suited to the task. Finally, the additional power and associated power regulator ensure that the power-hungry electromagnets do not affect the various timing mechanisms in use throughout the system, maintaining the accuracy of the measurement and feedback systems on board the refitted car.

# Hardware

The final hardware implementation of the system contains five main sub-systems; the front wheel steering column, the original circuitry connected to the DC motor (H-bridge), the power system, the optical mouse sensor SPI interface, and the serial communication system. Each of these sub-systems is connected to the refitted toy cars MSP430 Microcontroller which issues commands to the various sub-systems as it navigates through the course. The final hardware design is shown in Figure 7 below.

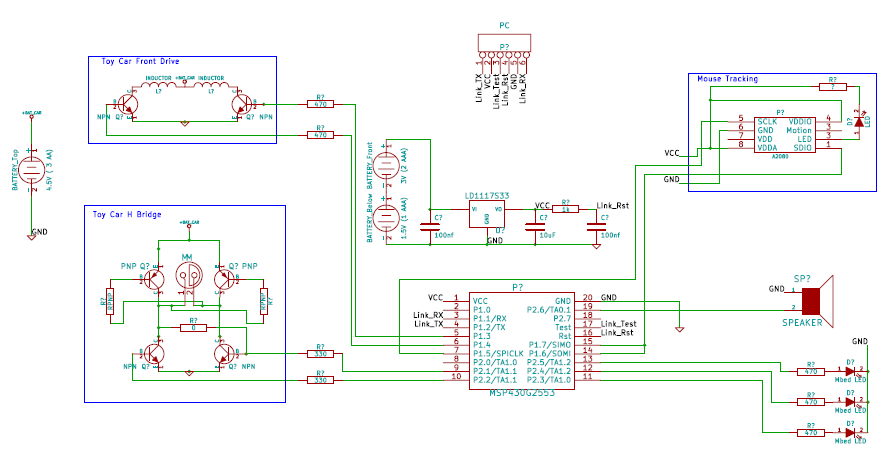


Figure 7 - Final Hardware circuit diagram.

## Steering System

As mentioned in the Design Specification section of this report, the steering system takes advantage of an electromagnet system to move the steering wheel to either the left or the right, depending on which electromagnet has been turned on. Figure 8 shows the circuit diagram of the steering system.

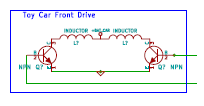


Figure 8 - Steering circuit diagram.

As you can see, two transistors on each side of the electromagnet series allow the current to flow through either the left or right electromagnet, depending on the control signals sent through the two wires connected to the microcontroller (via a resistor). When one of the transistors is actuated, one side of the circuit above is completed, allowing current to flow through one of the electromagnets. The regular magnet connected to the cars steering rack is then pulled in that direction, turning the wheels with it. By using this circuit that was already attached and connected to the various components on the car, two microcontroller pins can be simply connected in a regular output mode, and turned high when a specific direction of travel is required.

## Forward/Backwards System

Because the Toy Cars movement is controlled by a DC motor, a special type of circuit known as an H-bridge is needed to allow the wheels to turn both forward and backwards. A simplified H-bridge circuit is shown in Figure 9.

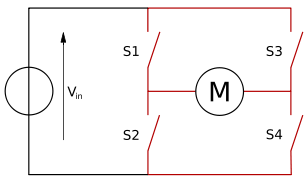


Figure 9 - Simplified H-bridge

As can be seen, by closing switches S2 and S3 at the same time, current would flow through the motor in a different direction from a situation when S1 and S4 were closed simultaneously. This allows a bi-directional device like a DC motor to turn in both a clockwise and anti-clockwise direction, meaning the car can travel both forwards and backwards. Other circuitry used to prevent short-circuits would also be connected, but is not shown in this diagram. The final circuit diagram for the system’s h-bridge is shown in Figure 10.

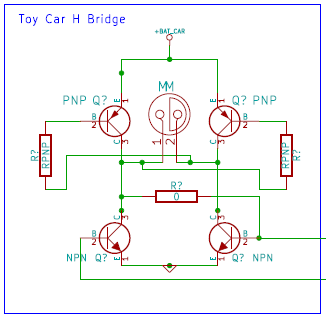


Figure 10 - The final system H-bridge

The circuit above shows the extra components required to prevent short circuits, as well as using transistors as switches. In this version of the H-bridge circuit, the second set of switches (S2 and S4 in the simplified circuit) is controlled by the state of the overall circuit to ensure that they cannot be both on simultaneously. Put simply, the circuit is wired in such a way that when S1 is closed, S4 is automatically closed and S2 is prevented from closing. The same occurs when S3 is closed – S2 is automatically closed, and S4 is prevented from closing.

## Power System

Because of the additional hardware attached to the refitted car, additional power is required. Three AA batteries (1.5V each, ~4.5V total) and two AAA batteries (1.5V each, ~4.5V total) were attached to the outside of the toy car to provide a larger power supply, as well as counterbalance some of the additional components such as the optical sensor. Because the new power supply is needed to not only drive and steer the toy car, it also needs to power the microcontroller, optical mouse sensor and serial data connection, the extra power provided by the additional batteries is required.

A voltage regulator is needed to ensure that new power supplies’ output is consistent. A LD1117S33 chip was used to filter out the noisy components from the output voltage, as well as regulate the voltage to a steady 3.3V output regardless of the current draw. The circuit diagram of the power supply with attached voltage regulator is shown in Figure 11. The capacitors in the circuit provide additional noise filtering and removal.

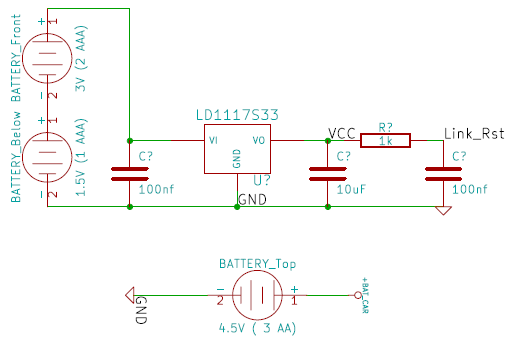


Figure 11 - Voltage regulator circuit.

## Location Sensor

The optical mouse sensor used for the location feedback is connected to the controlling microcontroller using an SPI (Serial Peripheral Interface) bus. SPI is a synchronous data link, although this system uses the three wire variant, employing a single bi-directional transmission line resulting in half-duplex communication.

The SPI uses three connections between the microcontroller (master) and the optical mouse sensor (slave) – the serial clock (SLCK) provided by the master device; the slave in, master out and slave out, master in connection (combined into one); and the slave select connection (also controlled by the master). At the optical mouse sensor, a small chip (A2080) accepts both the input and output serial connection on one pin that uses the SDIO interface to provide interrupts to the master microcontroller. The circuit located at the mouse sensor is shown in Figure 12.

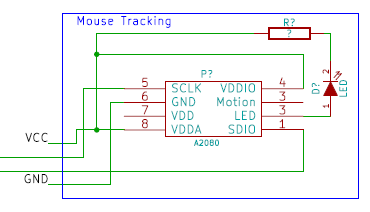


Figure 12 - The optical mouse sensor circuit

## Serial Communication System

The MSP430 G2553 in use in this system has a built in serial communication bus which implements a serial connection using Universal Asynchronous Receive Transmit (UART) and has one pin/wire for receive (RXD) and another to transmit (TXD). UART is a communication protocol used to message between two asynchronous devices. The G2553 uses a four pin configuration with the RX and TX pins used in addition to a Test and Reset pin, used to provide additional functionality. An example of this is shown in Figure 13.

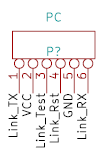


Figure 13 - Serial communication pins, this is the side of the connection connected to the host computer.

An advantage of the simplistic nature of UART is that it can be implemented wirelessly with relative ease. In this system, a wireless target board (RF2500) was connected to the serial connection of the microcontroller. This chip provides wireless serial communication over the 2.4GHz spectrum to a similar chip attached via USB to a host computer. Using a serial communication tool (like PuTTY) on the host computer, serial messages can be sent to the microcontroller controlling the toy car.

# Software Description

As was stated in the Project Specification the RC car is to be given two inputs, distance *d* and angle *a*, and travel for a total of 6 meters. There are several significant software challenges in this task, these are:

* Input : To give input to the MSP430 about the distance and angle
* Car Movements : To control the car’s movements i.e. forward, backward, left and right
* Mouse measurements : To receive input from the Mouse’s sensor and to gather the car’s actual position
* Path control : To ensure that the car follows the specified path, accounting for any drift that may be caused by external errors e.g the surface the car is travelling on

## Input to the car

The MSP430 has a serial port but cannot be accessed through the usual *scanf()* functions provided by C. Instead the UART chip, which Serial communication is reliant on, is communicated with directly using an open source library ‘uart\_io.c’ [1]. The two main methods in this library that are used are the *getc()* and the *getline()* methods.

*Getc()*  retrieves a character from the input stream or it will block until it receives one. *Getline()*, using the *getc()* method, reads a line from the input stream until a newline character (\n) is entered.

The *main()* function, the entry point of the system in main.c, contains a polling loop that listens for any serial input. Within the polling loop there is a switch statement which is dependent on the character entered. In the Final version of the code the character **‘p’** is the most important. When **‘p’** is entered the user is prompted to first insert the distance and then the angle and stores the values in global variables. To effectively make the car move, the speed should be set using the ‘s’character.

This is not the only use for the polling loop and switch statements. Although constructed with the intention of being used to input distance and angle values to the car it is also used as a Testing and Debugging mechanism. Table 1 describes each input character and its use. The Debug means that the car has to be flashed with the Debug constant defined for these options to be available.

Table 1: Serial character inputs

|  |  |  |
| --- | --- | --- |
| Input Character | Use | Mode |
| w | Moves Car Forward | Debug |
| s | Moves Car Backwards | Debug |
| a | Turns Car Left | Debug |
| d | Turns Car Right | Debug |
| + | Increases Car Speed | Normal |
| - | Decreases Car Speed | Normal |
| c | Set the Car speed | Normal |
| [space] | Stops The Car | Normal |
| h | High Resolution Mouse Sensor Input | Debug |
| l | Low Resolution Mouse Sensor Input | Debug |
| r | Reset All the position variables (angle, position) | Normal |
| p | Allow entering the distance and angle values | Normal |
| m | Music mode on! | Normal |

## **Car Movements**

Arguably the most important subtask of the project is to implement the software movement control of the car. The Hardware section describes how the cars drive mechanism is connected to the MSP430. Using this information the drive.h and drive.c file were created. These files contain the implementation of the functions used to control the movement and speed of the car.

The cars ability to move forward and backwards it dependant on the which bit is selected on port 2. The *forward()* method is used to make the car drive forward by selecting bit 1 on port 2 of the MSP430. The *backward()* method is used to make the car reverse by selecting bit 2 of the same port.  PWM is used to alter the speed at which the car is travelling at. The *speed()* method takes as input an *unsigned long* and sets the car to travel at the speed inputted.

Port 1 is connected to the cars steering mechanism. Using the *right()* method bit 3 is set on and using the *left()* method bit 4 is set on. It is important to note if the car is turning right a call to the method *left()* does not make the car turn left but it straightens the car up. A second call to the *left()* method is required to turn left. Similarly this is the case if car is to turn right when it is already turning left and is described by the state machine shown in Figure 14 .

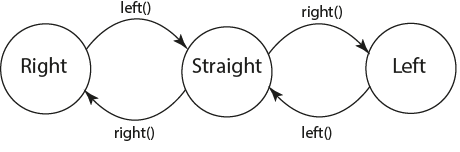


Figure 14 - State machine describing the turning mechanism

## Mouse Measurements

The mouse sensor used in this project outputs the change in distance in the x and y axis and will be used to calculate how far the car has travelled and measures the angle at which it is angled. It is connected to the MSP430 via the SPI bus. To communicate to the motion sensor on the mouse the spi.h and spi.c files were created. The methods *SPI\_Read() and SPI\_Write()* are used to read and write from the motion sensor in the mouse.  The methods *SPI\_WTW()*  takes care of the delays that are required between two successive write operations and *SPI\_WTR()*  is the delay required between a write and read operation. Both delays are defined in the datasheet [2]. Also defined in the datasheet are the addresses of the registers used to access the mouse sensor from the MSP430. The registers of interest are shown in Table 2.

Table 2: Mouse registers and their address

|  |  |
| --- | --- |
| Register | Address |
| Motion | 0x02 |
| Delta\_X | 0x03 |
| Delta\_Y | 0x04 |
| Data\_Out\_Upper | 0x0d |

In the *main()* method the motion register is first read to check whether or not the motion sensor has recorded motion. In the case that the motion sensor has new data, *SPI\_Read()* method reads of the Delta\_X and Delta\_Y registers. The *mx* and *my* variable are updated with the latest motion.

## Math Model of the System

The data retrieved from the motion sensor is vital to compute the distance the car has travelled and the current angle it has with respect to its starting position. This is clearly demonstrated in Figure 15.

θ represents the angle between the y axis of the base space and the y axis of the car space and is calculated using Equation 1 . This value is used to determine the angle that car has turned. dy and dx are the output from the motion sensors and mcc is the distance between the centre of the car and the centre of the motion sensor. X and Y are the coordinates of the centre of the car with respect to base space and are calculated using Equation 2 and 3

(1)

(2)

(3)

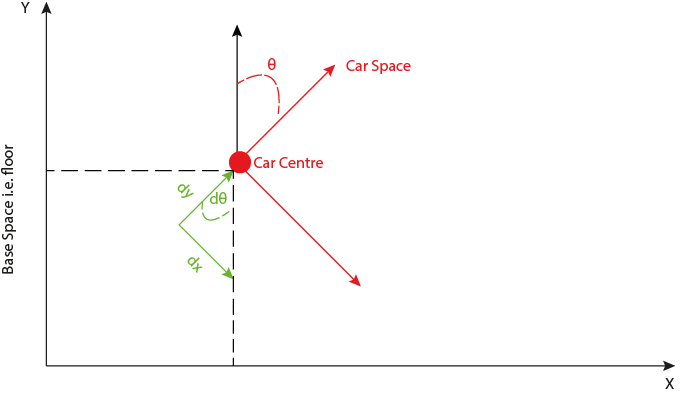


Figure 15 - A visual description of the math model

## Path Control

A combination of the work previously described in this section is put together to produce a mechanism to allow the car to complete the task defined in the Project Brief. A control flow diagram of the path control can be seen in Figure 16. Once the user inputs the distance and angle of the task in the system, the position of the car is updated as is the route to the destination. This is achieved by the *computePosition()* and the *postionControl()* methods, both of which are executed approximately every 10ms.

*ComputePosition()* takes as input the current Delta\_X and Delta\_Y register values obtained from the motion sensor. Then, it updates the angle, X and Y coordinates of the car with respect to the global coordinate system, using the Equations 1, 2 and 3 respectively.

*PositionControl()* is then called to compute how far the car is from its destination, and whether it has reached its destination or not so as to move the car appropriately. Due to the fact that the car is unreliable and can veer off course, a speed and angle control mechanism was incorporated into the *positionControl()* method. This method considers that the car has reached its destination as soon as the car is within 20cm radius from the destination point.

The *speedControl()* method is used to control the speed the car is travelling at in centimetres per second and ensures that the car is moving at this given speed no matter what surface it is on. The speed is calculated in the main function just after the update of the car’s position. Furthermore, the speed control is based on an integral action to compensate for the fact that the car will not move if the PWM power is not enough.

The *angleCorrection()* method takes as input the current angle of the car and a target angle. If the car’s current angle is within the range of the target angle ± a certain threshold there is no need for any corrective procedure. However, if the current angle lies out with this range the car will respond by appropriately turning left or right to correct any drift experienced by the car. The use of threshold prevents the car from turning every single time and makes its trajectory smoother.

The last detail for this explanation of the path control to be complete is the action of the main. When the distance and angle are inputted, the final coordinates [x, y] of the car are calculated using trigonometric maths. *PositionControl()* is then called to reach the turning point of the car whose coordinates are [0, distanceInputted]. Once the car arrived within 20cm of this point, the *PositionControl()* is called to reach the final coordinates. This 20cm margin allows preventing the case where the car would never exactly reach a precise point and thus it would turn around this point forever. Another effect of this margin is that the car will anticipate the turn it has to perform as it will begin to turn 20cm earlier than the inputted distance. Considering that the steering is quite slow, this is a good point.

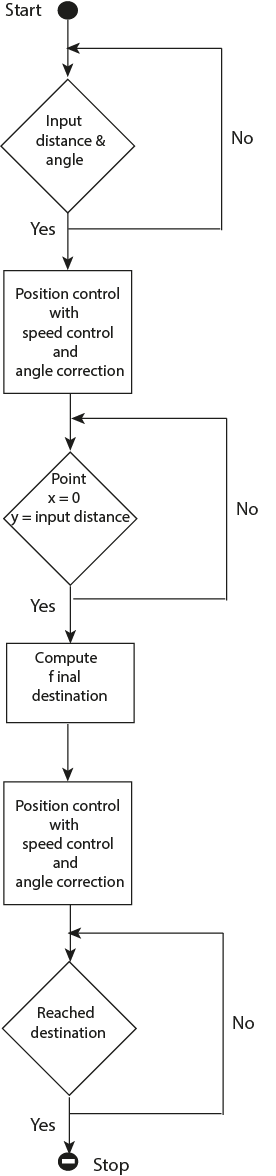


Figure 16 - Flow diagram of the path control

# Bibliography

|  |  |
| --- | --- |
| [1] | Alexander I. Mykyta, “UART Library,” 2012. |
| [2] | Avago Technologies, “ADNS-3080 DataSheet,” 2012. |