ISLAMIC UNIVERSITY OF TECHNOLOGY

ORGANISATION OF ISLAMIC COOPERATION

DEPARTMENT OF MECHANICAL AND CHEMICAL ENGINEERING

PROJECT THESIS PAPER

PROJECT TITLE:

"Trajectory Planning and Tracking of an Autonomous Robot Vehicle"

Submitted by:

Md Mahdiuzzahid Rifat (101406)

Md Raihanul Islam (101408)

Jonaed Bin Mustafa Kamal (101442)

Md Wasequl Islam (101444)

Project Supervisor:

Prof. Dr. Md. Nurul Absar Chowdhury

DECLARATION

This is to declare that the project titled *"Trajectory Planning and Tracking of an Autonomous Robot Vehicle"* was designed and successfully implemented by us under the supervision of Dr. Md. Nurul Absar Chowdhury, Professor, MCE Department, IUT. The following thesis has not been submitted elsewhere for the reward of any degree or diploma or for publication.

Dr. Md. Nurul Absar Chowdhury
Professor
MCE Department
Islamic University of Technology ……

Md Mahdiuzzahid Rifat
Student No. 101406 ……

Md Raihanul Islam
Student No. 101408 ……

Jonaed Bin Mustafa Kamal
Student No. 101442 ……

Md Wasequl Islam
Student No. 101444 ……

Acknowledgement

All praise to Allah. We wish to express our deep gratitude to our honorable project supervisor Professor Dr. Nurul Absar Chowdhury of IUT for his continuous supervision, guidance and suggestions in our project. Without his help this project would never have achieved this success.

Table of Contents

Abstract:

This paper is about trajectory planning and tracking of an autonomous robot vehicle. The autonomous robot was made by transforming a simple RC car into a GPS guided robot. Present and the target locations were given from a computer. Communication between the robot and the computer was done by using RF module. The receiver of the module was kept upon the car and the transmitter was connected to the computer. Tracking was done by the same GPS module that was used to guide the vehicle. All the processes were done using two Arduino Mega 2560 microprocessors. One was used for the transmitter and the other one was used for the receiver and control of the car.

Robot[1]:

Robot is an electro-mechanical device that can perform tasks by itself. It may require some to no human interaction. A robot requires an electric circuit and a programming to run. Robots range from humanoids to large industrial robots or even microscopic Nano robots. A robot contains sensors, control systems, manipulators, power supplies and software. All these systems work together (simultaneously or sequentially) to make a robot work. A robot maybe fully autonomous or semi-autonomous. A fully autonomous robot can work without the help of any human interaction.

Characteristics of a Robot:

- **Sensing**

  First of all a robot must have the capabilities of sensing its environment. Sensing is done in the same way that a human senses its surroundings. While a human senses its environment with the help of ears, eyes, nose a robot senses its surroundings using different sensors. Different sensors are used for a robot to sense its surroundings. Some of the sensors are-

  - Sonar
  - Electro-magnetic
  - Touch
  - Chemical
  - Altitude
  - Thermal

- **Movement**

  A robot needs to move around. Movement can be done by wheels, legs, propellers or by some other means. A robot may be able to travel from one place to another some of its parts may move keeping the robot in a specific place.

- **Energy**

  Power is required for the robot to perform its tasks. The robot may power itself or it may require the help of human to power it up. The robot may be electrically powered, battery powered or solar powered. The amount of power required and the way it needs to be powered depends on the tasks that the robot has to perform.

[1] Most of the descriptive parts (definitions, descriptions, history etc.) of our paper were taken from different websites and papers and are not the outcomes of our project.

- **Intelligence**

  A robot also needs intelligence to perform the task. Robot intelligence is created by programming. A coding is needed to run the robot and make it understand what it needs to do.

Autonomous Robot:

An autonomous robot is a robot that can perform tasks completely on its own or sometimes with some help from human. These types of robots require very little amount of human help or sometimes no human help at all. A very common example of autonomous robot is Sony's **AIBO** series. These were pet toys resembling dogs, cats or other pets. These autonomous robots were made mainly for entertainment but could do a lot of work completely on its own.

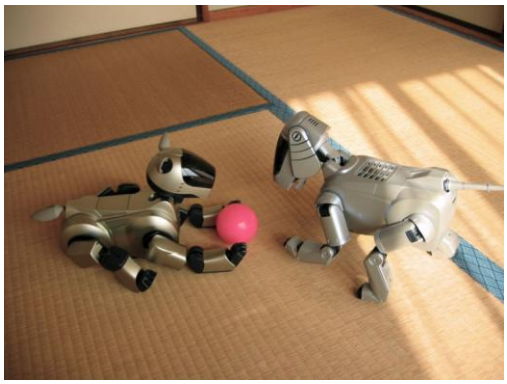


Figure 1: Sony's AIBOs- a great example of autonomous robots

Qualities of an Autonomous Robot:

Self-maintenance:

The first requirement for a robot to be autonomous is the ability to take care of it itself. For example, if an autonomous robot needs to recharge itself, it should be able to go to a charging dock by itself.

Self maintenance is done by using some proprioceptive sensors. Proprioceptive sensors are sensors which help the robot to gain information about its internals. Such as if anything inside the robot goes wrong it should be able to sense that and alert the user of it.

Common proprioceptive sensors are:

1. Thermal
2. Hall Effect
3. Optical
4. Contact

Sensing the environment

Next it should have the ability to sense its surroundings. These robots sense their surroundings and act by themselves. This does not need any human interaction. Sensing environment is done with the help of exteroceptive sensors.

Common exteroceptive sensors are:

1. Electromagnetic spectrum
2. Sound
3. Touch
4. Chemical sensors

Task performance

The next step in autonomous behavior is to actually perform a physical task. Autonomous task performance requires a robot to perform conditional tasks. For instance, security robots can be programmed to detect intruders and respond in a particular way depending upon where the intruder is.

Indoor position sensing and navigation

For a robot to associate behaviors with a place (localization) requires it to know where it is and to be able to navigate point-to-point. At first, autonomous navigation was based on planar sensors, such as laser range-finders, that can only sense at one level. The most advanced systems now fuse information from various sensors for both localization (position) and navigation.

Outdoor autonomous position-sensing and navigation

Outdoor autonomy is most easily achieved in the air, since obstacles are rare. Outdoor autonomy is most difficult for ground vehicles, due to:

a) 3-dimensional terrain

b) Great disparities in surface density

c) Weather exigencies and

d) Instability of the sensed environment

Short History of Robot:

Remote-controlled systems

Remotely operated vehicles were demonstrated in the late 19th Century in the form of several types of remotely controlled torpedoes. The early 1870s saw remotely controlled torpedoes by John Ericsson (pneumatic), John Louis Lay (electric wire guided), and Victor von Scheliha (electric wire guided).

The Brennan torpedo, invented by Louis Brennan in 1877 was powered by two contra-rotating propellers that were spun by rapidly pulling out wires from drums wound inside the torpedo.

Differential speed on the wires connected to the shore station allowed the torpedo to be guided to its target, making it "the world's first *practical* guided missile". In 1897 the British inventor Ernest Wilson was granted a patent for a torpedo remotely controlled by "Hertzian" (radio) waves and in 1898 Nikola Tesla publicly demonstrated a wireless-controlled torpedo that he hoped to sell to the US Navy.

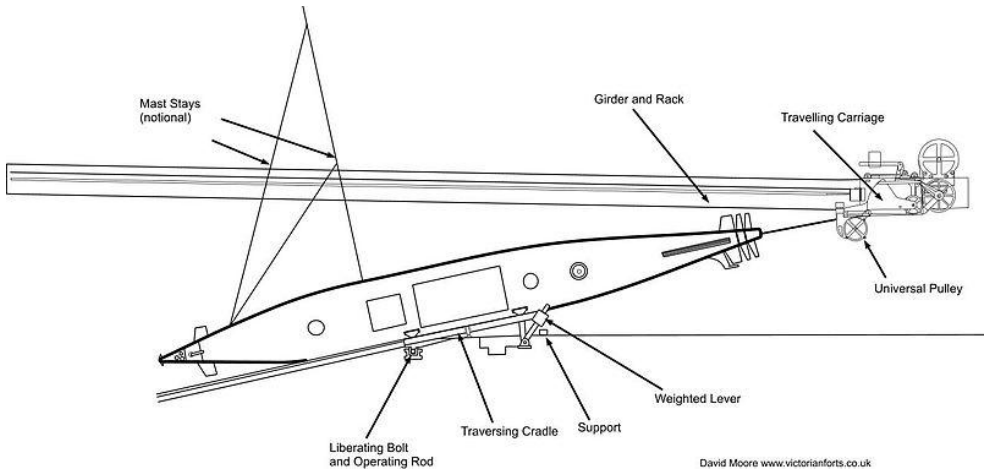


Figure 2: Brennan Torpedo

Archibald Low, known as the "father of radio guidance systems" for his pioneering work on guided rockets and planes during the First World War. In 1917, he demonstrated a remote controlled aircraft to the Royal Flying Corps and in the same year built the first wire-guided rocket

Modern autonomous robots

The first electronic autonomous robots with complex behavior were created by William Grey Walter of the Burden Neurological Institute at Bristol, England in 1948 and 1949. His first robots, named *Elmer* and *Elsie*, were constructed between 1948 and 1949 and were often described as *tortoises* due to their shape and slow rate of movement. The three-wheeled tortoise robots were capable of photo taxis, by which they could find their way to a recharging station when they ran low on battery power.

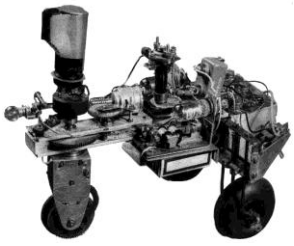


Figure 3: Elsie

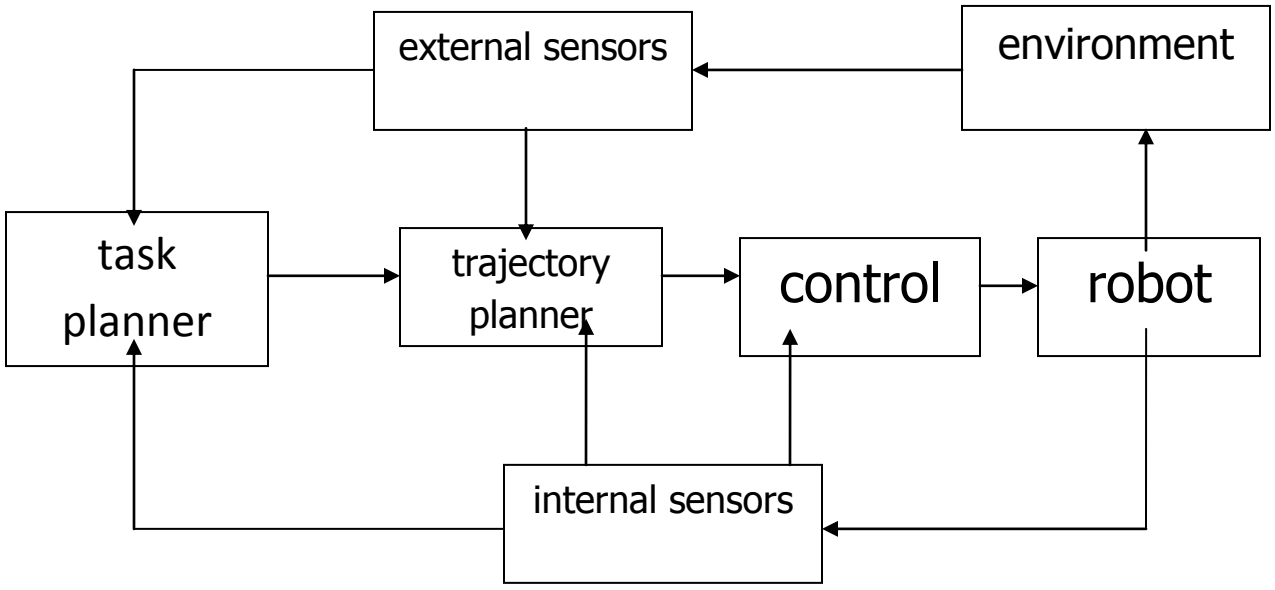
The first digitally operated and programmable robot was invented by George Devol in 1954 and was ultimately called the Unimate. This ultimately laid the foundations of the modern robotics industry which could lift hot pieces of metal from a die casting machine and stack them.

The first palletizing robot was introduced in 1963 by the Fuji Yusoki Kogyo Company. In 1973, a robot with six electromechanically driven axes was patented by KUKA robotics in Germany, and the programmable universal manipulation arm was invented by Victor Scheinman in 1976, and the design was sold to Unimation.

Commercial and industrial robots are now in widespread use performing jobs more cheaply or with greater accuracy and reliability than humans. They are also employed for jobs which are too dirty, dangerous or dull to be suitable for humans. Robots are widely used in manufacturing, assembly and packing, transport, earth and space exploration, surgery, weaponry, laboratory research, and mass production of consumer and industrial goods.

Figure 4: A Modern Autonomous Robot

Manufacturing a Robot:



1. At first the task of the robot is selected. The robot can perform one task only or several tasks. For example, the robot can be an automated hand for collecting samples. In this case collecting samples is the selected task.
2. Then the path of the robot or the trajectory is selected. That is the robot will follow a specific path in order to complete the task. For the above example the hand should move between the samples and the collecting box. The path maybe one-dimensional, two-dimensional or three-dimensional.
3. A control system is then designed which will control the robot that is controlling all of its movements. The control system includes a computer system. The computer system maybe a fully fledged computer or it may be a micro-processor.
4. Necessary sensors and tracking devices are required to sense the robot's internal and external states and to track its path.
5. The computer system and the sensors and tracking devices are then assembled together into a body. This will complete the construction of the robot.
6. The robot will be able to move according to its path using the external and internal sensors.
7. Some internal sensors must be used to monitor the control system e.g. battery level.
8. If the robot does not act to the desired task, then the control system and the sensors will have to be revised.

Autonomous Vehicle:

An **autonomous vehicle** can be defined as a **driverless vehicle** or **robot vehicle**. An autonomous vehicle is a vehicle that can travel by itself. It not only fulfills transportation capabilities of a traditional vehicle but also can drive on its own. It has the ability of a complete autonomous robot. Autonomous vehicles sense their surroundings with the help of radar, GPS, or computer vision. It has the ability to avoid obstacles in its path and move according to the set path and destination. Some autonomous vehicles can update their maps based on sensory input, allowing the vehicles to keep track of their position even when conditions change or when they enter uncharted environments.

Benefits of Autonomous Vehicle:

- No human error will result in significant decrease in road accidents.
- It will be much helpful for disabled people as there will be no need to actually drive the car.
- Traffic congestion will also significantly decrease.
- Traffic flow will be much faster and smoother.
- Occupants of the vehicle can spend time in doing other things without driving.

- The vehicle will be able to drive in uncertain terrains and make the occupant know where it won't be able to go.
- Research can be done in places where it is very unsafe for human to go.

Drawbacks:

However there are several possible drawbacks which are listed below-

- The automated vehicle might not go to places where it was possible for the vehicle to go in case of human driving.
- Use of this technology must require a standard rule for the whole world, otherwise different rules in different countries might result in complete prohibition of the technology.
- Responsibilities of the manufacturers will increase as car manufacturer will be responsible for any car crash.

Trajectory Planning[2]

Introduction

Trajectory is defined as the path that a moving object follows through space as a function of time. Trajectory planning can be defined as the planning of the trajectory that the object will follow. This will include setting up the start and end points as well as designing the path that the object will follow.

Criteria for Trajectories:
• Efficient - easy to compute and execute
• Predictable and accurate - should not degenerate near a singularity
• Position, Velocity and Acceleration should be smooth functions of time
• The output of the trajectory planner is a sequence of arm configurations (either in joint or Cartesian space) that form the input to the feedback control system of the robot arm.
While this problem seems fairly straight forward at first glance, it is not. Complications include:
• Hand rotation also needs to be planned and controlled (e.g. last 3 joints of a 6-axis robot like a puma)

[2] The **trajectory planning** segment is taken from **reference 13** and not an outcome of our project

• Trajectory planning is often kinematic analysis only; the actual dynamics of the robot (acceleration of the link mass, friction, gravity, gear backlash etc are ignored.

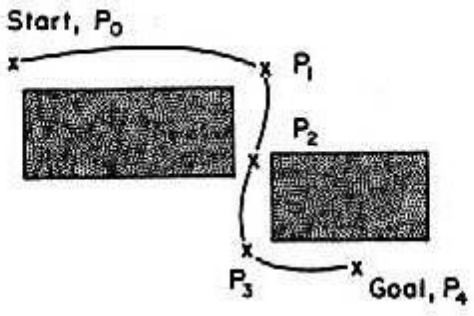


Figure 5: Trajectory planning of a robot

Ways of Planning a Trajectory:

Trajectory planning can be done by several methods. Trajectory planning can be done using-

1. Joint space
2. Cartesian space
3. Polynomial interpolation etc.

Procedures of Trajectory Planning:

Typically trajectory planning includes the following procedures-

1. Defining Cartesian pose points (position+ orientation)
2. Programming a velocity (average) between these points as 0-100%, 100% being the maximum system value.
3. Linear interpolation in the joint space between points samples from the built trajectory.

Path Planning

Path planning is a purely geometric matter, since it implies the generation of a geometric path without a specified time law, while the trajectory planning assigns time to the geometric path.

Path planning is one of the most interesting topics of advanced robotics. The challenge of path planning for a mobile system is to find a collision free motion between the initial and final position. The easiest method is to plan the path in a known and in a static environment.

There are three most important approaches to path planning: road-map, cell decomposition and potential field

The road-map approach is based on the capture of the free space connectivity on a system of 1-dimensional curves (road-map) in the C-free space or in its closure. The so-built road-map is used as a set of standardized paths. The path planning is then reduced to linking the initial and final configurations to, with the aim to find a path in between the two configurations.

The cell decomposition methods divide the free space of the robot in regions, called cells, so that it is possible to easily create a path between any two configurations of the same cell. Afterwards a connectivity graph is constructed, which represents the adjacency relations between cells.

A different approach consists of the discretization of the C-space in a dense and regular grid of configurations, finding a free path. As the grid is generally very wide, this approach requires very powerful heuristics used in the path research. An alternative approach that has achieved remarkable results in extremely complex problems is the probabilistic roadmap planners (PRMs). It is a methodology that uses probabilistic techniques (random sampling) to construct the roadmap. The great advantage of PRM is that their complexity depends more on the path complexity rather than the complexity of the environment and the size of the configurations space.

Trajectory Planning

The target of the trajectory planning is to generate the reference inputs for the manipulator control system that ensures the implementation of the desired movement. It can be assumed that a trajectory planning algorithm takes as inputs the geometric path, the kinematic and dynamic constraints of the manipulator; the output is the trajectory of the joints, or of the end-effector, expressed as a sequence of values of position, velocity and acceleration.

The geometric path is usually specified in the operative space, since both the task to perform and the obstacles to avoid can be more naturally described in this space. The trajectory planning in the operative space consists to generate a time sequence of values (within the constraints imposed) that specify the position and orientation of the end-effector. This solution is adopted when the movement is on a path with geometric characteristics defined in the operative space; the path can be specified exactly by primitive path or in an approximated way by the allocation of path points that are usually connected with polynomial sequences. Anyway, since the control action on the manipulator is made on the joints, a kinematic inversion is necessary in order to derive the evolution of variables in the joint space.

A trajectory planned in the joint space consists in the acquisition of the values, for each joint of the manipulator, corresponding to the via-points set by the user. The trajectories defined in the joint space are generated by means of interpolation functions which respect the limits imposed.

This planning solution can also easily avoid the problems involved in moving near singular configurations and the possible presence of redundant degrees of mobility.

The main drawback is related to the fact that the execution of a movement planned in the joint space is not so easy to predict in the operative space, due to the nonlinear effects introduced by the direct kinematics. Regardless of the particular strategy adopted, it is essential that the laws of motion generated in the planning phase are such as induce forces and torques at the joints compatible with the given constraints, hence reducing the possibility to excite mechanical resonance modes, that are most often not modeled. Starting from this fundamental consideration, it is necessary that the planning algorithms output smooth trajectories, i.e. trajectories with a high order of continuity.

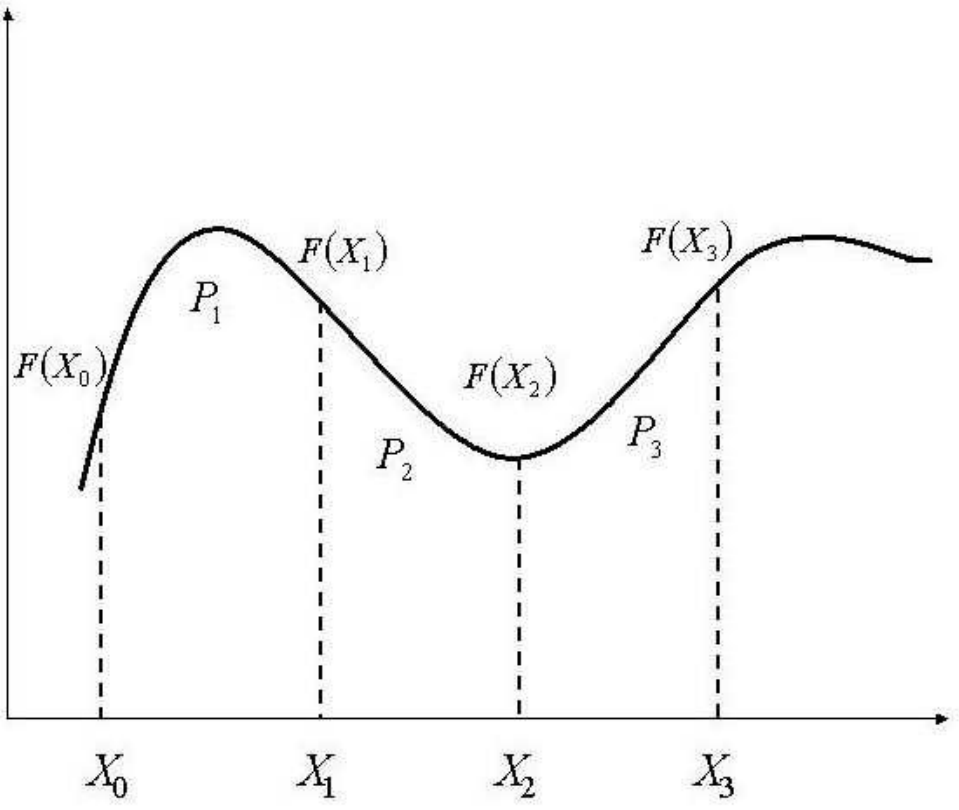
In the scientific literature on the trajectory planning problem it is possible to find different optimality criteria; the most significant are:

• Minimum execution time;

• Minimum energy (or actuator effort);

• Minimum jerk.

Kinematics of Trajectory Planning:



We can design the trajectory of a robot by dividing the path into very small parts e.g. P_1, P_2, P_3 etc. P_1 has the starting point coordinates x_0 and $F(x_0)$ and end points x_1 and $F(x_1)$. Similarly for P_2 we get the points x_1, $F(x_1)$ and x_2, $F(x_2)$. In this way we can obtain a number of paths P_1, P_2, P_3 etc. By adding the paths we can obtain the whole path. The accuracy of the path depends on the number of divisions of the path.

Vehicle Kinematics Analysis

Under the basic assumptions of planar motion, rigid body and non-slippage of tire, the large size vehicle with four steering wheels as shown in figure 1 can be approximated using a bicycle model. To describe the vehicle motion a global co-ordinate X-Y is fixed on the horizontal plane on which the vehicle moves.

Reference point C is chosen at the center of gravity of the vehicle body. Its coordinates (X,Y) represents the position of the vehicle; Vehicle velocity v is defined at the reference point C; Heading angle Ψ is the angle from the X-axis to the longitudinal axis of the vehicle body AB; Course angle Υ s the angle from the X-axis to the direction of the vehicle velocity, v; Slip-side Angle β is the angle from the longitudinal axis of the vehicle body AB to the direction of vehicle velocity, v; Turning radius r is the distance between the reference point C and the Instant Rotating Center (IRC) O; Front Wheel Velocity V_f is the velocity defined at the intersection of the mid-plane of the virtual front wheel and the velocity defined at the intersection of the mid-plane of the virtual front wheel and the front wheel axle, A; Rear Wheel Velocity v_r is the velocity defined at the intersection of the mid plane of the virtual rear wheel and the rear wheel axle, B; Front wheel steering angle δ_f is the angle from the longitudinal axis of the vehicle body AB to the direction of v_f Rear wheel steering angle δ_r is the angle from the longitudinal axis of the vehicle body AB to the direction of v_r.

Referring to figure 2, the kinematic model of 4WS vehicles can be expressed as follows

$$\dot{X} = vcos(\Psi + \beta) \qquad (1\text{-}1)$$

$$\dot{Y} = v\sin(\Psi + \beta) \qquad (1\text{-}2)$$

$$\dot{\Psi} = \frac{vcos\beta(tan_f + tan_r)}{I_f + I_r} \qquad (1\text{-}3)$$

Where

$$B = \arctan\frac{I_f \cos\delta_r + I_r \cos\delta_r}{I_f + I_r} \quad (1\text{-}4)$$

And

$$V = \frac{v_f \cos\delta_f + v_r \cos\delta_r}{I_f + I_r} \quad (1\text{-}5)$$

Communication Mediums of Autonomous Car:

An autonomous car communicates using different techniques. Most common are-

- GPS
- Radar
- Computer Vision etc.

For our communication system we used GPS.

GPS:

The **Global Positioning System** (**GPS**) is a space-based satellite navigation system that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

The GPS project was developed in 1973 to overcome the limitations of previous navigation systems, integrating ideas from several predecessors, including a number of classified engineering design studies from the 1960s. GPS was created and realized by the U.S. Department of Defense (DoD) and was originally run with 24 satellites. It became fully operational in 1995. Advances in technology and new demands on the existing system have now led to efforts to modernize the GPS system and implement the next generation of GPS III satellites and Next Generation Operational Control System (OCX).

Basic concept of GPS

A GPS receiver calculates its position by precisely timing the signals sent by GPS satellites high above the Earth. Each satellite continually transmits messages that include:

- the time the message was transmitted and,
- satellite position at time of message transmission.

The receiver uses the messages it receives to determine the transit time of each message and computes the distance to each satellite using the speed of light. Each of these distances and satellites' locations defines a sphere. The receiver is on the surface of each of these spheres when the distances and the satellites' locations are correct. These distances and satellites' locations are used to compute the location of the receiver using the navigation equations. This location is then displayed, perhaps with a moving map display or latitude and longitude; elevation or altitude information may be included, based on height above the geode.

Figure 6: GPS Block II-F satellite in Earth orbit

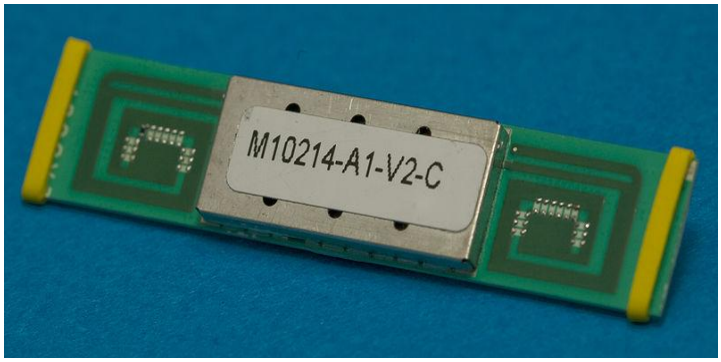


Figure 6: A typical GPS receiver with integrated antenna

In typical GPS operation, four or more satellites must be visible to obtain an accurate result. Four sphere surfaces typically do not intersect. Because of this, it can be said with confidence that when the navigation equations are solved to find an intersection, this solution gives the position of the receiver along with the difference between the time kept by the receiver's on-board clock and the true time-of-day.

GPS Tracking:

A **GPS tracking** unit is a device that uses the Global Positioning System to determine the precise location of a vehicle, person, or other asset to which it is attached and to record the position of the asset at regular intervals. The recorded location data can be stored within the tracking unit, or it may be transmitted to a central location data base, or internet-connected computer, using a cellular (GPRS or SMS), radio, or satellite modem embedded in the unit. This allows the asset's location to be displayed against a map backdrop either in real time or when analyzing the track later, using GPS tracking software.

Architecture of a GPS Tracker:

A GPS tracker essentially contains GPS module to receive the GPS signal and calculate the coordinates. For data loggers it contains large memory to store the coordinates, data pushers additionally contains the GSM/GPRS modem to transmit this information to a central computer either via SMS or via GPRS in form of IP packets.

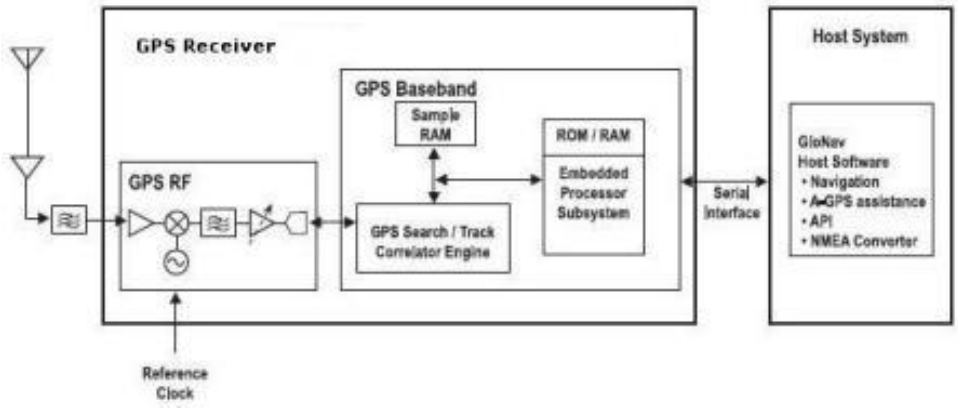


Figure 7: Architecture of a GPS tracker

Types of GPS trackers

Usually, a GPS tracker will fall into one of these three categories.

Data loggers

A GPS logger simply logs the position of the device at regular intervals in its internal memory. Modern GPS loggers have either a memory card slot, or internal flash memory and a USB port. Some act as a USB flash drive. This allows downloading of the track log data for further analyzing in a computer.

Figure 8: Typical data logger

Data pushers

Data pusher is the most common type of GPS tracking unit, used for asset tracking, personal tracking and Vehicle tracking system.

Also known as a *GPS beacon*, this kind of device pushes or sends the position of the device as well as other information like speed or altitude at regular intervals, to a determined server that can store and instantly analyze the data.

A GPS navigation device and a mobile phone sit side-by-side in the same box, powered by the same battery. At regular intervals, the phone sends a text message via SMS or GPRS, containing the data from the GPS receiver.

Most 21st-century GPS trackers provide data "push" technology, enabling sophisticated GPS tracking in business environments, specifically organizations that employ a mobile workforce, such as a commercial fleet. Typical GPS tracking systems used in commercial fleet management have two core parts: location hardware (or tracking device) and tracking software. This combination is often referred to as an "Automatic Vehicle Location" system. The tracking device is most often hardware installed in the vehicle.

Data pullers

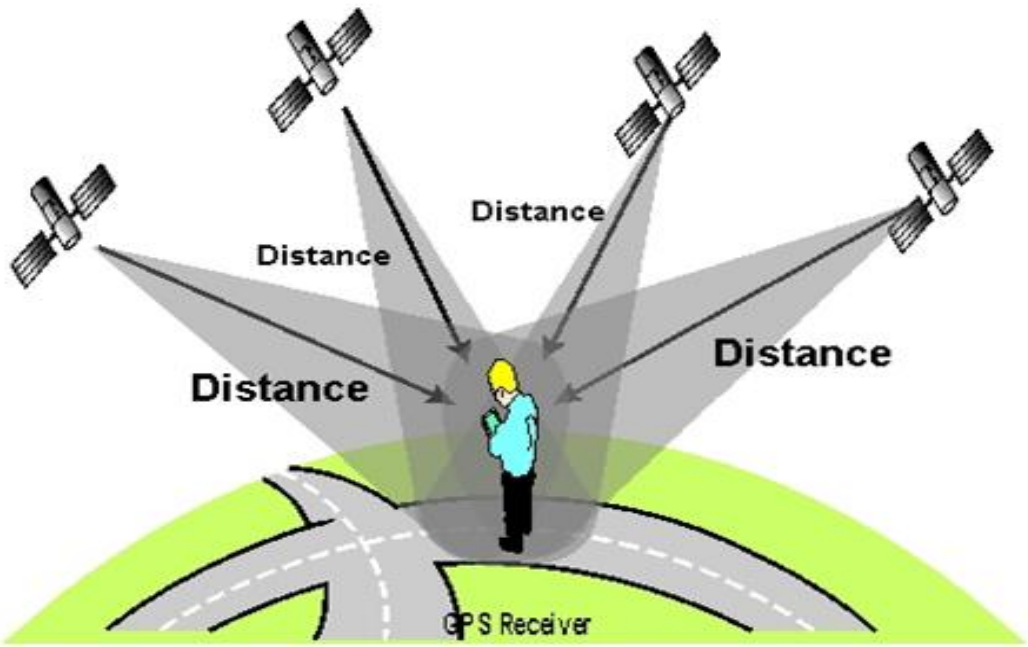
GPS data pullers are also known as *GPS transponders*. Contrary to data pushers, that send the position of the devices at regular intervals (push technology), these devices are always-on and can be queried as often as required (pull technology). This technology is not in widespread use, but an example of this kind of device is a computer connected to the Internet and running gpsd.

These can often be used in the case where the location of the tracker will only need to be known occasionally e.g. placed in property that may be stolen, or that does not have constant source of energy to send data on a regular basis, like freights or containers.

Data Pullers are coming into more common usage in the form of devices containing a GPS receiver and a cell phone which, when sent a special SMS message reply to the message with their location.

GPS Technique:

The GPS position of an object can be found by the following technique.



At least four GPS satellites are needed to determine one object's location. Data from four GPS satellites intersect at one point to determine the objects location. The more the satellites are used, the more accurate position can be found.

GPS Tracking Method:

A GPS tracking system uses the Global Navigation Satellite System (GNSS) network. This network incorporates a range of satellites that use microwave signals that are transmitted to GPS devices to give information on location, vehicle speed, time and direction. So, a GPS tracking system can potentially give both real-time and historic navigation data on any kind of journey.

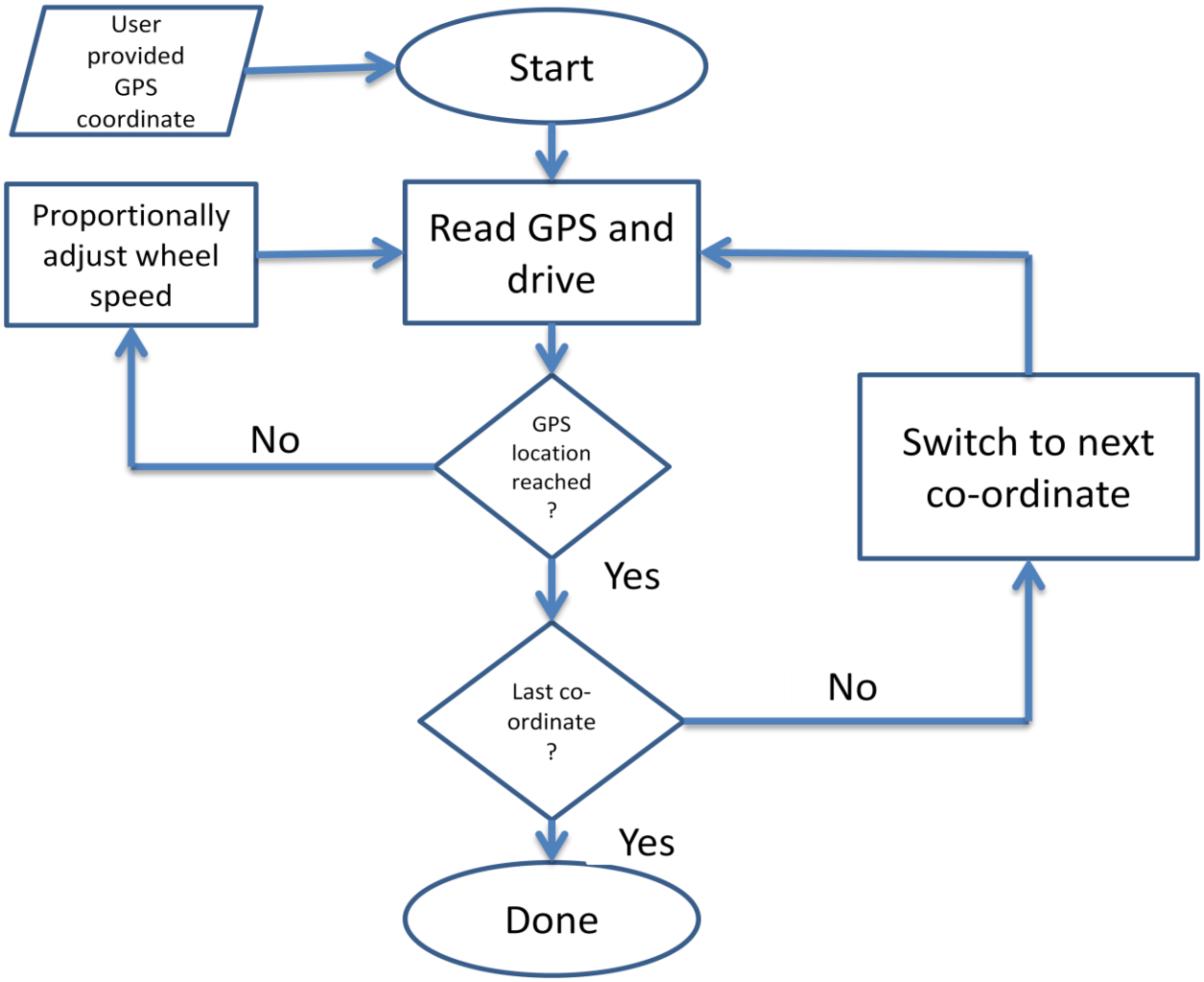
The method is given below:

1. GPS provides special satellite signals, which are processed by a receiver.
2. These GPS receivers not only track the exact location but can also compute velocity and time. The positions can even be computed in three-dimensional views with the help of four GPS satellite signals. The Space Segment of the Global Positioning System consists of 27 Earth-orbiting GPS satellites. There are 24 operational and 3 extra (in case one fails) satellites that move round the Earth each 12 hours and send radio signals from space that are received by the GPS receiver.

3. The control of the Positioning System consists of different tracking stations that are located across the globe. These monitoring stations help in tracking signals from the GPS satellites that are continuously orbiting the earth.
4. Space vehicles transmit microwave carrier signals.
5. The users of Global Positioning Systems have GPS receivers that convert these satellite signals so that one can estimate the actual position, velocity and time.

GPS Tracking Method of a Robot:

The GPS tracking method of a robot can be described by the following diagram:



1. The user provides a GPS coordinate to the robot.
2. The robot reads the GPS coordinate and moves to the specified location.
3. It then reads the GPS coordinate of its present point.
4. If the robot reads the coordinate as the end point it stops.
5. Otherwise it adjusts its speed and wheel direction and moves to the next coordinate.

This process is done over and over again until the robot reaches its destination. This is a very effective way of tracking the path that it follows.

Microprocessor:

A microprocessor is a device that works as a computer's central processing (CPU) unit but is constructed on a single integrated circuit (IC) or sometimes on a few integrated circuits. The microprocessor is a multipurpose, programmable device that accepts digital data as input, processes it according to instructions stored in its memory, and provides results as output. It is an example of sequential digital logic, as it has internal memory. Microprocessors operate on numbers and symbols represented in the binary numeral system.

Advantages of Using a Microprocessor over a whole CPU:

- Reduces cost
- Reduces power consumption
- Reduces size
- Decreases electrical connections thus making more reliable
- Less prone to damage
- Can be used in very small devices
- Reduces computer size

Before microprocessors, small computers had been implemented using racks of circuit boards with many medium and small scale integrated circuits. Microprocessors integrated this into one or a few large-scale ICs. Continued increases in microprocessor capacity have since rendered other forms of computers almost completely obsolete.

Structure:

The internal structure of a microprocessor varies depending upon the intended purpose. A minimal microprocessor might only include an arithmetic logic unit (ALU) and a control logic section. The ALU performs operations such as addition, subtraction, and operations such as AND or OR. The logic section retrieves instruction operation codes from memory, and initiates whatever sequence of operations of the ALU requires carrying out the instruction. A single operation code might affect many individual data paths, registers, and other elements of the

processor. However design of a practical microprocessor is much more complex. For example the internal structure of the Arduino mega microprocessor is shown in the diagram below.

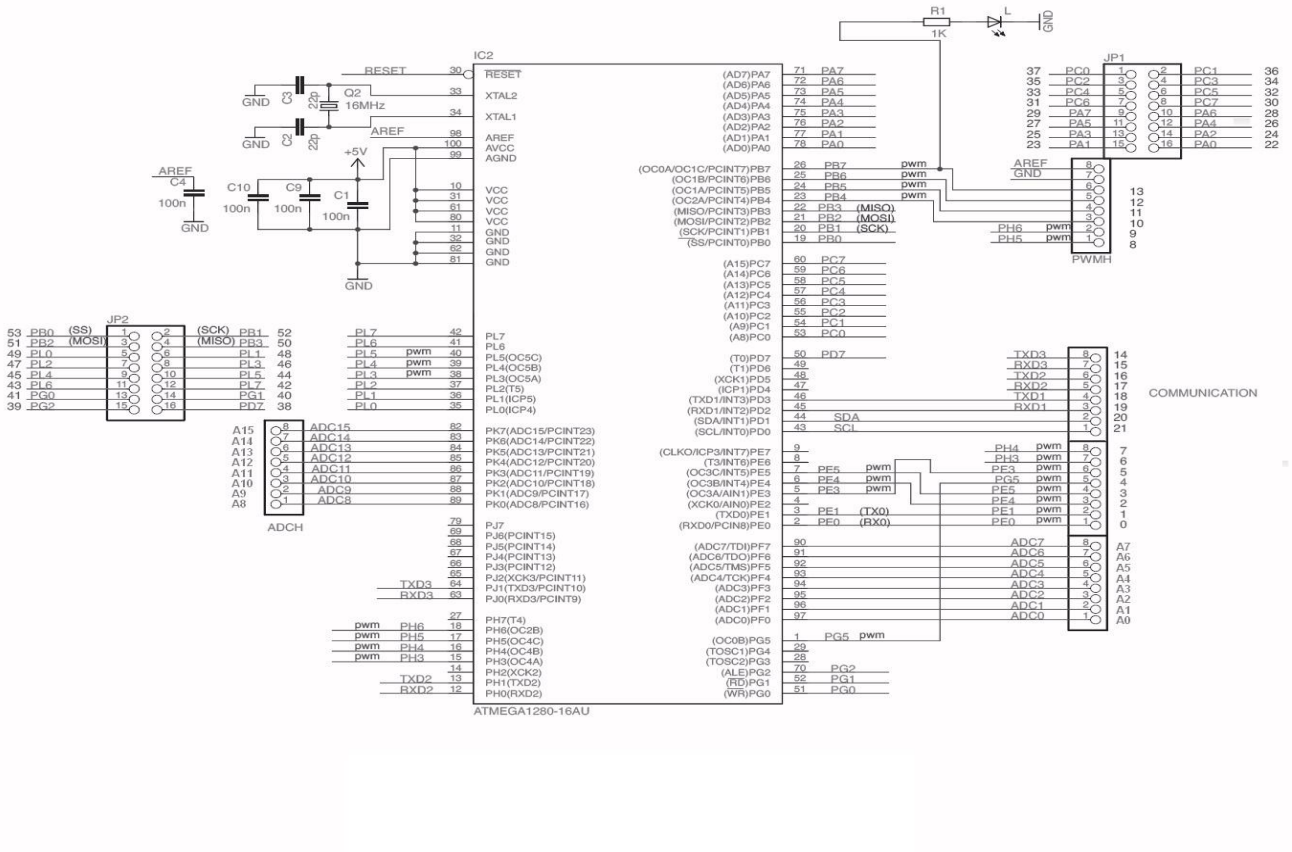
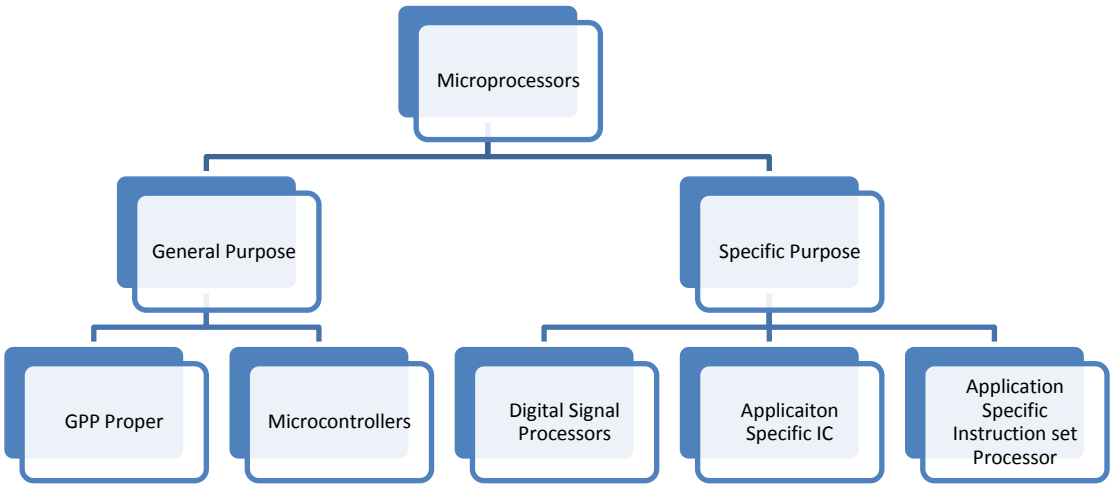


Figure 9: Internal structure of the Arduino mega

Classification of Microprocessor:



Some Microprocessors Used in Small Robots:

1. Arduino UNO
2. Arduino Duemilanove
3. Arduino Nano
4. Arduino Mega
5. Arduino Mega 2560
6. Arduino Pro Mini 328
7. Lynx motion Mini Atom
8. DFRobot Romeo etc.

Microcontroller

A microcontroller is a small computer on a single integrated circuit containing a processor core, memory and programmable input/output peripherals. Program memory in the form of NOR flash or OTP TOM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications.

Some microcontrollers may use four-bit words and operate at clock rate frequencies as low as 4 kHz, for low power consumption (single-digit mill watts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just Nano watts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.

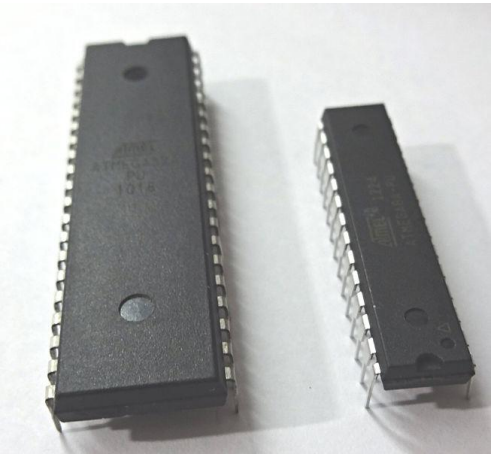


Figure: AT mega 32 bit and 8 bit microcontroller

Features

- A true computer on a chip
- It can function as complete system without any external component
- Flexibility
- Embedded applications
- Application software is ROM-based
- On-chip Oscillator
- On-chip RAM of limited size
- On-chip ROM of limited size
- On-chip Input output ports
- Serial port is also built in
- Timer counter is also on-chip

History

The first microprocessor was the 4-bit Intel 4004 released in 1971, with the Intel 8008 and other more capable microprocessors becoming available over the next several years. However, both processors required external chips to implement a working system. The Smithsonian Institution says TI engineers Gary Boone and Michael Cochran succeeded in creating the first microcontroller in 1971. The result of their work was the TMS 1000, which became commercially available in 1974. It combined read-only memory, read/write memory, processor and clock on one chip and was targeted at embedded systems.

Partly in response to the existence of the single-chip TMS 1000, Intel developed a computer system on a chip optimized for control applications, the Intel 8048, with commercial parts first shipping in 1977. It combined RAM and ROM on the same chip. This chip would find its way into over one billion PC keyboards, and other numerous applications.

Most microcontrollers at this time had two variants. One had an erasable EPROM program memory, with a transparent quartz window in the lid of the package to allow it to be erased by exposure to ultraviolet light. The other was a PROM variant which was only programmable once.

In 1993, the introduction of EEPROM memory allowed microcontrollers (beginning with the Microchip PIC16x84) to be electrically erased quickly without an expensive package as required for EPROM, allowing both rapid prototyping, and In System Programming. (EEPROM technology had been available prior to this time, but the earlier EEPROM was more expensive and less durable, making it unsuitable for low-cost mass-produced microcontrollers.) The same year, Atmel introduced the first microcontroller using Flash memory, a special type of EEPROM. Other companies rapidly followed suit, with both memory types.

Embedded Design

A micro controller is a self-contained system with a processor, memory and other peripherals which can be used as an embedded system. Although embedded systems are sophisticated, they require minimal memory and program length, which make them easier platform to work with. Typical input output device s such as switch, relay, leds, LCD display, radio frequency devices can be used.

Microcontrollers are capable of providing real time response to events in the embedded system they are working. When a certain event occur, an interrupt system can signal the processor to suspend processing the current instruction sequence and to begin an interrupt service routine (ISR, or "interrupt handler"). The ISR will perform any processing required based on the source of the interrupt, before returning to the original instruction sequence. Possible interrupt sources are device dependent, and often include events such as an internal timer overflow, completing an analog to digital conversion.

Interrupts

Micro controllers must provide real time (predictable, though not necessarily fast) response to events in the embedded system they are controlling. When certain events occur, an interrupt system can signal the processor to suspend processing the current instruction sequence and to begin an interrupt service routine (ISR, or "interrupt handler"). The ISR will perform any processing required based on the source of the interrupt, before returning to the original instruction sequence. Possible interrupt sources are device dependent, and often include events such as an internal timer overflow, completing an analog to digital conversion, a logic level change on an input such as from a button being pressed, and data received on a communication link. Where power consumption is important as in battery operated devices, interrupts may also wake a microcontroller from a low power sleep state where the processor is halted until required to do something by a peripheral event.

The Structure of an interrupt is given below:

- Used for real time application
- Marking feature
- Enabling Interrupt and disabling interrupt
- Switching (context)
- Vectored interrupt
- Priority

Interrupt latency

In contrast to general-purpose computers, microcontrollers used in embedded systems often seek to optimize interrupt latency over instruction throughput. Issues include both reducing the latency, and making it be more predictable (to support real-time control).

When an electronic device causes an interrupt, the intermediate results (registers) have to be saved before the software responsible for handling the interrupt can run. They must also be restored after that software is finished. If there are more registers, this saving and restoring process takes more time, increasing the latency. Ways to reduce such context/restore latency include having relatively few registers in their central processing units (undesirable because it slows down most non-interrupt processing substantially), or at least having the hardware not save them all (this fails if the software then needs to compensate by saving the rest "manually"). Another technique involves spending silicon gates on "shadow registers": One or more duplicate registers used only by the interrupt software, perhaps supporting a dedicated stack.

Other factors affecting interrupt latency include:

- Cycles needed to complete current CPU activities. To minimize those costs, microcontrollers tend to have short pipelines (often three instructions or less), small write buffers, and ensure that longer instructions are continual or restartable. Reduced instruction set computing/RISC design principles ensure that most instructions take the same number of cycles, helping avoid the need for most such continuation/restart logic.
- The length of any critical section that needs to be interrupted. Entry to a critical section restricts concurrent data structure access. When a data structure must be accessed by an interrupt handler, the critical section must block that interrupt. Accordingly, interrupt latency is increased by however long that interrupt is blocked. When there are hard external constraints on system latency, developers often need tools to measure interrupt latencies and track down which critical sections cause slowdowns.
    - One common technique just blocks all interrupts for the duration of the critical section. This is easy to implement, but sometimes critical sections get uncomfortably long.
    - A more complex technique just blocks the interrupts that may trigger access to that data structure. This is often based on interrupt priorities, which tend to not correspond well to the relevant system data structures. Accordingly, this technique is used mostly in very constrained environments.
    - Processors may have hardware support for some critical sections. Examples include supporting atomic access to bits or bytes within a word, or other atomic access primitives like the Load-link/store-conditional/LDREX/STREX exclusive access primitives introduced in the ARMv6 architecture.
- Interrupt nesting. Some microcontrollers allow higher priority interrupts to interrupt lower priority ones. This allows software to manage latency by giving time-critical interrupts higher priority (and thus lower and more predictable latency) than less-critical ones.
- Trigger rate. When interrupts occur back-to-back, microcontrollers may avoid an extra context save/restore cycle by a form of tail call optimization.

Lower end microcontrollers tend to support fewer interrupt latency controls than higher end ones.

Program

A typical microcontroller program must fit in the available on-chip program memory, as it is costly to provide a system with external and expandable memory. Compilers and assemblers convert high-level language and assembly language codes into a compact machine code the microcontroller's memory. Depending on the device, the program memory may be permanent, read-only memory that can only be programmed at the factory, or program memory that may be field-alterable flash or erasable read-only memory.

There are other versions available where the ROM is accessed as an external device rather than as internal memory, however these are becoming increasingly rare due to the widespread availability of cheap microcontroller programmers.

A customizable microcontroller incorporates a block of digital logic that can be personalized in order to provide additional processing capability, peripherals and interfaces that are adapted to the requirements of the application. For example, the AT91CAP from Atmel has a block of logic that can be customized during manufacture according to user requirements.

Other microcontroller features

Microcontrollers usually contain from several to dozens of general purpose input/output pins (GPIO). GPIO pins are software configurable to either an input or an output state. When GPIO pins are configured to an input state, they are often used to read sensors or external signals. Configured to the output state, GPIO pins can drive external devices such as LEDs or motors.

Many embedded systems need to read sensors that produce analog signals. This is the purpose of the analog-to-digital converter (ADC). Since processors are built to interpret and process digital data, i.e. 1s and 0s, they are not able to do anything with the analog signals that may be sent to it by a device. So the analog to digital converter is used to convert the incoming data into a form that the processor can recognize. A less common feature on some microcontrollers is a digital-to-analog converter (DAC) that allows the processor to output analog signals or voltage levels.

In addition to the converters, many embedded microprocessors include a variety of timers as well. One of the most common types of timers is the Programmable Interval Timer (PIT). A PIT may either count down from some value to zero, or up to the capacity of the count register, overflowing to zero. Once it reaches zero, it sends an interrupt to the processor indicating that it has finished counting. This is useful for devices such as thermostats, which periodically test the temperature around them to see if they need to turn the air conditioner on, the heater on, etc.

A dedicated Pulse Width Modulation (PWM) block makes it possible for the CPU to control power converters, resistive loads, motors, etc., without using lots of CPU resources in tight timer loops.

A micro-controller is a single integrated circuit, commonly with the following features:

- central processing unit - ranging from small and simple 4-bit processors to complex 32- or 64-bit processors
- volatile memory (RAM) for data storage
- ROM, EPROM, EEPROM or Flash memory for program and operating parameter storage
- discrete input and output bits, allowing control or detection of the logic state of an individual package pin
- serial input/output such as serial ports (UARTs)
- other serial communications interfaces like I²C, Serial Peripheral Interface and Controller Area Network for system interconnect
- peripherals such as timers, event counters, PWM generators, and watchdog
- clock generator - often an oscillator for a quartz timing crystal, resonator or RC circuit
- many include analog-to-digital converters, some include digital-to-analog converters
- in-circuit programming and debugging support

This integration drastically reduces the number of chips and the amount of wiring and circuit board space that would be needed to produce equivalent systems using separate chips. Furthermore, on low pin count devices in particular, each pin may interface to several internal peripherals, with the pin function selected by software. This allows a part to be used in a wider variety of applications than if pins had dedicated functions.

Micro-controllers have proved to be highly popular in embedded systems since their introduction in the 1970s.

Some microcontrollers use a Harvard architecture: separate memory buses for instructions and data, allowing accesses to take place concurrently. Where a Harvard architecture is used, instruction words for the processor may be a different bit size than the length of internal memory and registers; for example: 12-bit instructions used with 8-bit data registers.

The decision of which peripheral to integrate is often difficult. The microcontroller vendors often trade operating frequencies and system design flexibility against time-to-market requirements from their customers and overall lower system cost. Manufacturers have to balance the need to minimize the chip size against additional functionality.

Microcontroller architectures vary widely. Some designs include general-purpose microprocessor cores, with one or more ROM, RAM, or I/O functions integrated onto the package. Other designs are purpose built for control applications. A micro-controller instruction set usually has many instructions intended for bit-wise operations to make control programs more compact.[9] For example, a general purpose processor might require several instructions to

test a bit in a register and branch if the bit is set, where a micro-controller could have a single instruction to provide that commonly required function.

Microcontrollers typically do not have a math coprocessor, so floating point arithmetic is performed by software.

Programming environments

Microcontrollers were originally programmed only in assembly language, but various high-level programming languages are now also in common use to target microcontrollers. These languages are either designed specially for the purpose, or versions of general purpose languages such as the C programming language. Compilers for general purpose languages will typically have some restrictions as well as enhancements to better support the unique characteristics of microcontrollers. Some microcontrollers have environments to aid developing certain types of applications. Microcontroller vendors often make tools freely available to make it easier to adopt their hardware.

Many microcontrollers are so quirky that they effectively require their own non-standard dialects of C, such as SDCC for the 8051, which prevent using standard tools (such as code libraries or static analysis tools) even for code unrelated to hardware features. Interpreters are often used to hide such low level quirks.

Interpreter firmware is also available for some microcontrollers. For example, BASIC on the early microcontrollers Intel 8052;[10] BASIC and FORTH on the Zilog Z8[11] as well as some modern devices. Typically these interpreters support interactive programming.

Simulators are available for some microcontrollers. These allow a developer to analyze what the behavior of the microcontroller and their program should be if they were using the actual part. A simulator will show the internal processor state and also that of the outputs, as well as allowing input signals to be generated. While on the one hand most simulators will be limited from being unable to simulate much other hardware in a system, they can exercise conditions that may otherwise be hard to reproduce at will in the physical implementation, and can be the quickest way to debug and analyze problems.

Recent microcontrollers are often integrated with on-chip debug circuitry that when accessed by an in-circuit emulator via JTAG, allow debugging of the firmware with a debugger.

As of 2008 there are several dozen microcontroller architectures and vendors including:

- ARM core processors (many vendors)
    - ARM Cortex-M cores are specifically targeted towards microcontroller applications
- Atmel AVR (8-bit), AVR32 (32-bit), and AT91SAM (32-bit)
- Cypress Semiconductor's M8C Core used in their PSoC (Programmable System-on-Chip)
- Freescale ColdFire (32-bit) and S08 (8-bit)
- Freescale 68HC11 (8-bit)

- Intel 8051
- Infineon: 8-bit XC800, 16-bit XE166, 32-bit XMC4000 (ARM based Cortex M4F), 32-bit TriCore and, 32-bit Aurix Tricore Bit microcontrollers[12]
- MIPS
- Microchip Technology PIC, (8-bit PIC16, PIC18, 16-bit dsPIC33 / PIC24), (32-bit PIC32)
- NXP Semiconductors LPC1000, LPC2000, LPC3000, LPC4000 (32-bit), LPC900, LPC700 (8-bit)
- Parallax Propeller
- PowerPC ISE
- Rabbit 2000 (8-bit)
- Renesas Electronics: RL78 16-bit MCU; RX 32-bit MCU; SuperH; V850 32-bit MCU; H8; R8C 16-bit MCU
- Silicon Laboratories Pipelined 8-bit 8051 Microcontrollers and mixed-signal ARM-based 32-bit microcontrollers
- STMicroelectronics STM8 (8-bit), ST10 (16-bit) and STM32 (32-bit)
- Texas Instruments TI MSP430 (16-bit) C2000 (32-bit)
- Toshiba TLCS-870 (8-bit/16-bit).

Many others exist, some of which are used in very narrow range of applications or are more like applications processors than microcontrollers. The microcontroller market is extremely fragmented, with numerous vendors, technologies, and markets. Note that many vendors sell or have sold multiple architectures.

Microcontroller embedded memory technology

Since the emergence of microcontrollers, many different memory technologies have been used. Almost all microcontrollers have at least two different kinds of memory, a non-volatile memory for storing firmware and a read-write memory for temporary data.

Data

From the earliest microcontrollers to today, six-transistor SRAM is almost always used as the read/write working memory, with a few more transistors per bit used in the register file. FRAM or MRAM could potentially replace it as it is 4 to 10 times denser which would make it more cost effective.

In addition to the SRAM, some microcontrollers also have internal EEPROM for data storage; and even ones that do not have any (or not enough) are often connected to external serial EEPROM chip (such as the BASIC Stamp) or external serial flash memory chip.

A few recent microcontrollers beginning in 2003 have "self-programmable" flash memory.[3]

Firmware

The earliest microcontrollers used mask ROM to store firmware. Later microcontrollers (such as the early versions of the Freescale 68HC11 and early PIC microcontrollers) had quartz windows that allowed ultraviolet light in to erase the EPROM.

The Microchip PIC16C84, introduced in 1993,[13] was the first microcontroller to use EEPROM to store firmware. In the same year, Atmel introduced the first microcontroller using NOR Flash memory to store firmware.[3]

Arduino Mega 2560[3]

Arduino Mega is a microcontroller board based on the ATmega2560. It has 54 digital I/O pins (14 of them are PWM outputs), 16 analog inputs, a 16 MHz oscillator, a USB connection and a Power jack. Arduino Mega 2560 can be powered by connecting it to a laptop via a USB cable or by powering it with a battery.

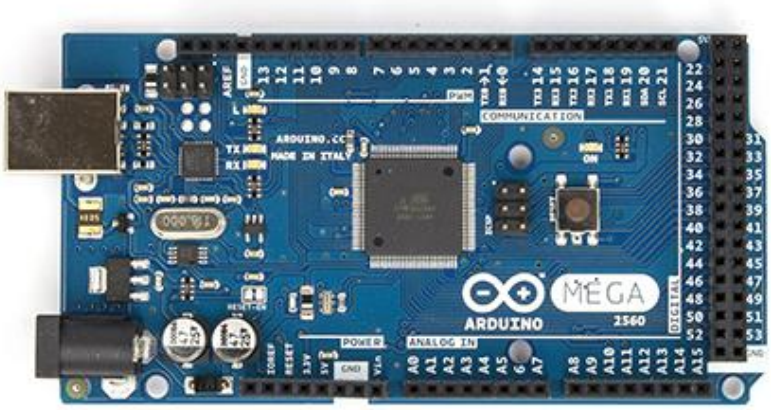


Figure: Front Side of Arduino Mega 2560

[3] Descriptions and specifications of Arduino and Atmega are taken from **reference 32**

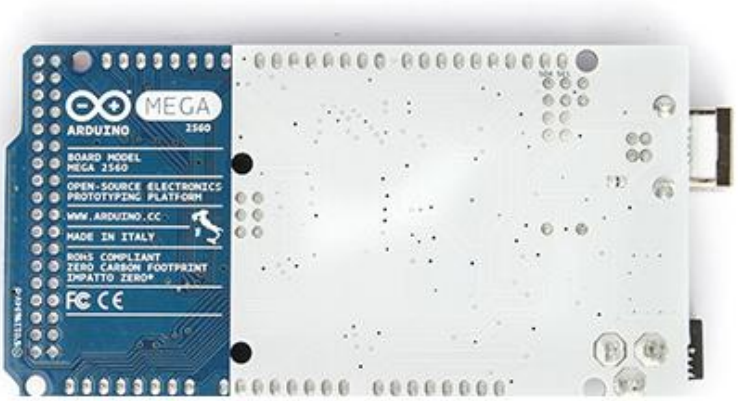


Figure: Back Side of Arduino Mega 2560

Specification:

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by boot loader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

AT Mega 2560:

The microcontroller that is used in the Arduino 2560 is the AT Mega 2560. It is a small, compact, high-performance and low power microcontroller used to make small robots.

Specifications:

- 8-bit AVR RISC-based microcontroller
- 256KB ISP flash memory
- 8KB SRAM
- 4KB EEPROM
- 86 general purpose I/O lines
- 32 general purpose working registers
- Real-time counter
- Six flexible timer/counters with compare modes
- PWM
- 4 USARTs
- Byte oriented 2-wire serial interface
- 16-channel 10-bit A/D converter
- JTAG interface for on-chip debugging
- 16 MIPS at 16 MHz
- Operates between 4.5-5.5 volts.

Figure: ATMega2560 microcontroller

Power

Arduino Mega 2560 can be powered using an external power supply or via the USB connection. The external power source can either be selected from an AC to DC adapter or battery. The board has 5 power pins which can supply power as well:

- **VIN** – The input voltage to the Arduino board when it's using an external power source The power can be supplied by a pin or through the power jack.

- **5V** – This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V).
- **3V3 -** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND - Ground Pins**

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the boot loader), 8 KB of SRAM and 4 KB of EEPROM

Input and Output

All 54 digital pins on the Mega can be used as an input or output, using the pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can supply or receive a maximum of 40 m. Besides some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 2 to 13 and 44 to 46.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **TWI: 20 (SDA) and 21 (SCL).** Support TWI communication using the Wire library. Note that these pins are not in the same location as the TWI pins on the Duemilanove or Diecimila.

The Mega2560 has 16 analog inputs, each of which provides 10 bits of resolution.

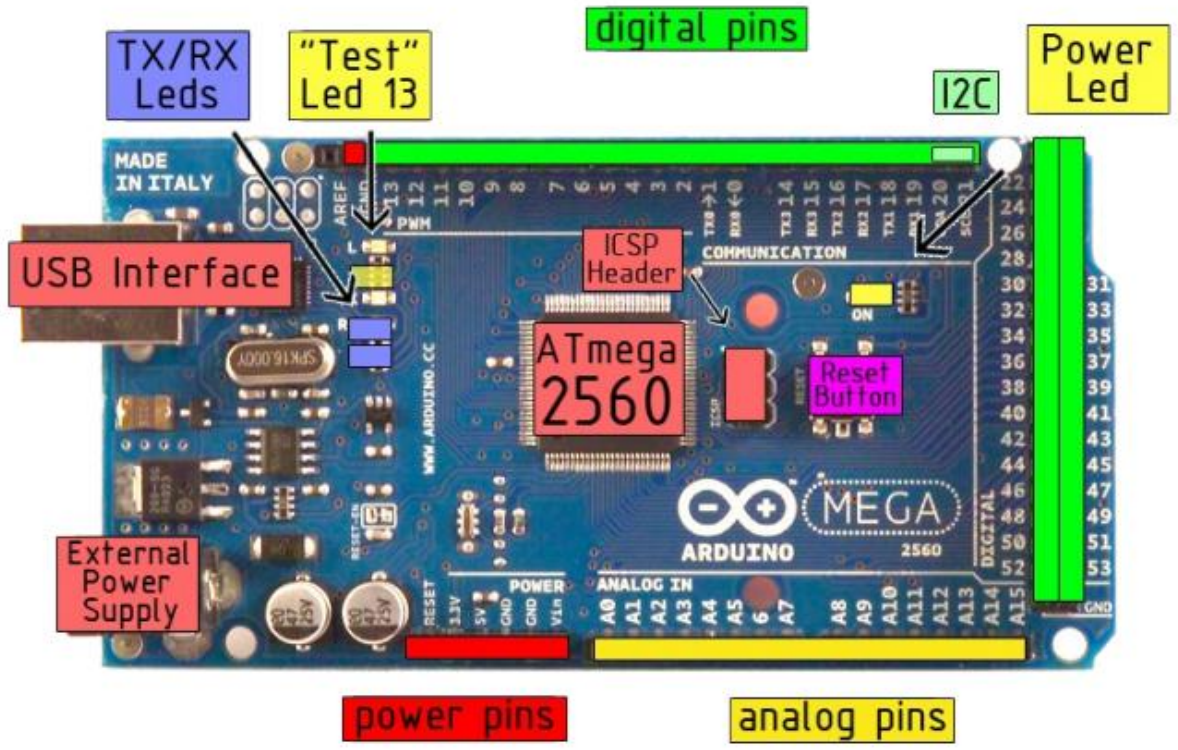


Figure: Arduino Mega 2560 Pin description

The PIN mapping of Arduino Mega 2560 is given below:

Arduino Mega 2560 PIN mapping table

Pin Number	Pin Name	Mapped Pin Name
1	PG5 (OC0B)	Digital pin 4 (PWM)
2	PE0 (RXD0/PCINT8)	Digital pin 0 (RX0)
3	PE1 (TXD0)	Digital pin 1 (TX0)
4	PE2 (XCK0/AIN0)	
5	PE3 (OC3A/AIN1)	Digital pin 5 (PWM)
6	PE4 (OC3B/INT4)	Digital pin 2 (PWM)
7	PE5 (OC3C/INT5)	Digital pin 3 (PWM)
8	PE6 (T3/INT6)	
9	PE7 (CLKO/ICP3/INT7)	
10	VCC	VCC
11	GND	GND

12	PH0 (RXD2)	Digital pin 17 (RX2)
13	PH1 (TXD2)	Digital pin 16 (TX2)
14	PH2 (XCK2)	
15	PH3 (OC4A)	Digital pin 6 (PWM)
16	PH4 (OC4B)	Digital pin 7 (PWM)
17	PH5 (OC4C)	Digital pin 8 (PWM)
18	PH6 (OC2B)	Digital pin 9 (PWM)
19	PB0 (SS/PCINT0)	Digital pin 53 (SS)
20	PB1 (SCK/PCINT1)	Digital pin 52 (SCK)
21	PB2 (MOSI/PCINT2)	Digital pin 51 (MOSI)
22	PB3 (MISO/PCINT3)	Digital pin 50 (MISO)
23	PB4 (OC2A/PCINT4)	Digital pin 10 (PWM)
24	PB5 (OC1A/PCINT5)	Digital pin 11 (PWM)
25	PB6 (OC1B/PCINT6)	Digital pin 12 (PWM)
26	PB7 (OC0A/OC1C/PCINT7)	Digital pin 13 (PWM)
27	PH7 (T4)	
28	PG3 (TOSC2)	
29	PG4 (TOSC1)	
30	RESET	RESET
31	VCC	VCC
32	GND	GND
33	XTAL2	XTAL2
34	XTAL1	XTAL1
35	PL0 (ICP4)	Digital pin 49
36	PL1 (ICP5)	Digital pin 48
37	PL2 (T5)	Digital pin 47
38	PL3 (OC5A)	Digital pin 46 (PWM)
39	PL4 (OC5B)	Digital pin 45 (PWM)
40	PL5 (OC5C)	Digital pin 44 (PWM)
41	PL6	Digital pin 43
42	PL7	Digital pin 42
43	PD0 (SCL/INT0)	Digital pin 21 (SCL)
44	PD1 (SDA/INT1)	Digital pin 20 (SDA)
45	PD2 (RXDI/INT2)	Digital pin 19 (RX1)
46	PD3 (TXD1/INT3)	Digital pin 18 (TX1)

47	PD4 (ICP1)	
48	PD5 (XCK1)	
49	PD6 (T1)	
50	PD7 (T0)	Digital pin 38
51	PG0 (WR)	Digital pin 41
52	PG1 (RD)	Digital pin 40
53	PC0 (A8)	Digital pin 37
54	PC1 (A9)	Digital pin 36
55	PC2 (A10)	Digital pin 35
56	PC3 (A11)	Digital pin 34
57	PC4 (A12)	Digital pin 33
58	PC5 (A13)	Digital pin 32
59	PC6 (A14)	Digital pin 31
60	PC7 (A15)	Digital pin 30
61	VCC	VCC
62	GND	GND
63	PJ0 (RXD3/PCINT9)	Digital pin 15 (RX3)
64	PJ1 (TXD3/PCINT10)	Digital pin 14 (TX3)
65	PJ2 (XCK3/PCINT11)	
66	PJ3 (PCINT12)	
67	PJ4 (PCINT13)	
68	PJ5 (PCINT14)	
69	PJ6 (PCINT 15)	
70	PG2 (ALE)	Digital pin 39
71	PA7 (AD7)	Digital pin 29
72	PA6 (AD6)	Digital pin 28
73	PA5 (AD5)	Digital pin 27
74	PA4 (AD4)	Digital pin 26
75	PA3 (AD3)	Digital pin 25
76	PA2 (AD2)	Digital pin 24
77	PA1 (AD1)	Digital pin 23
78	PA0 (AD0)	Digital pin 22
79	PJ7	
80	VCC	VCC
81	GND	GND

82	PK7 (ADC15/PCINT23)	Analog pin 15
83	PK6 (ADC14/PCINT22)	Analog pin 14
84	PK5 (ADC13/PCINT21)	Analog pin 13
85	PK4 (ADC12/PCINT20)	Analog pin 12
86	PK3 (ADC11/PCINT19)	Analog pin 11
87	PK2 (ADC10/PCINT18)	Analog pin 10
88	PK1 (ADC9/PCINT17)	Analog pin 9
89	PK0 (ADC8/PCINT16)	Analog pin 8
90	PF7 (ADC7)	Analog pin 7
91	PF6 (ADC6)	Analog pin 6
92	PF5 (ADC5/TMS)	Analog pin 5
93	PF4 (ADC4/TMK)	Analog pin 4
94	PF3 (ADC3)	Analog pin 3
95	PF2 (ADC2)	Analog pin 2
96	PF1 (ADC1)	Analog pin 1
97	PF0 (ADC0)	Analog pin 0
98	AREF	Analog Reference
99	GND	GND

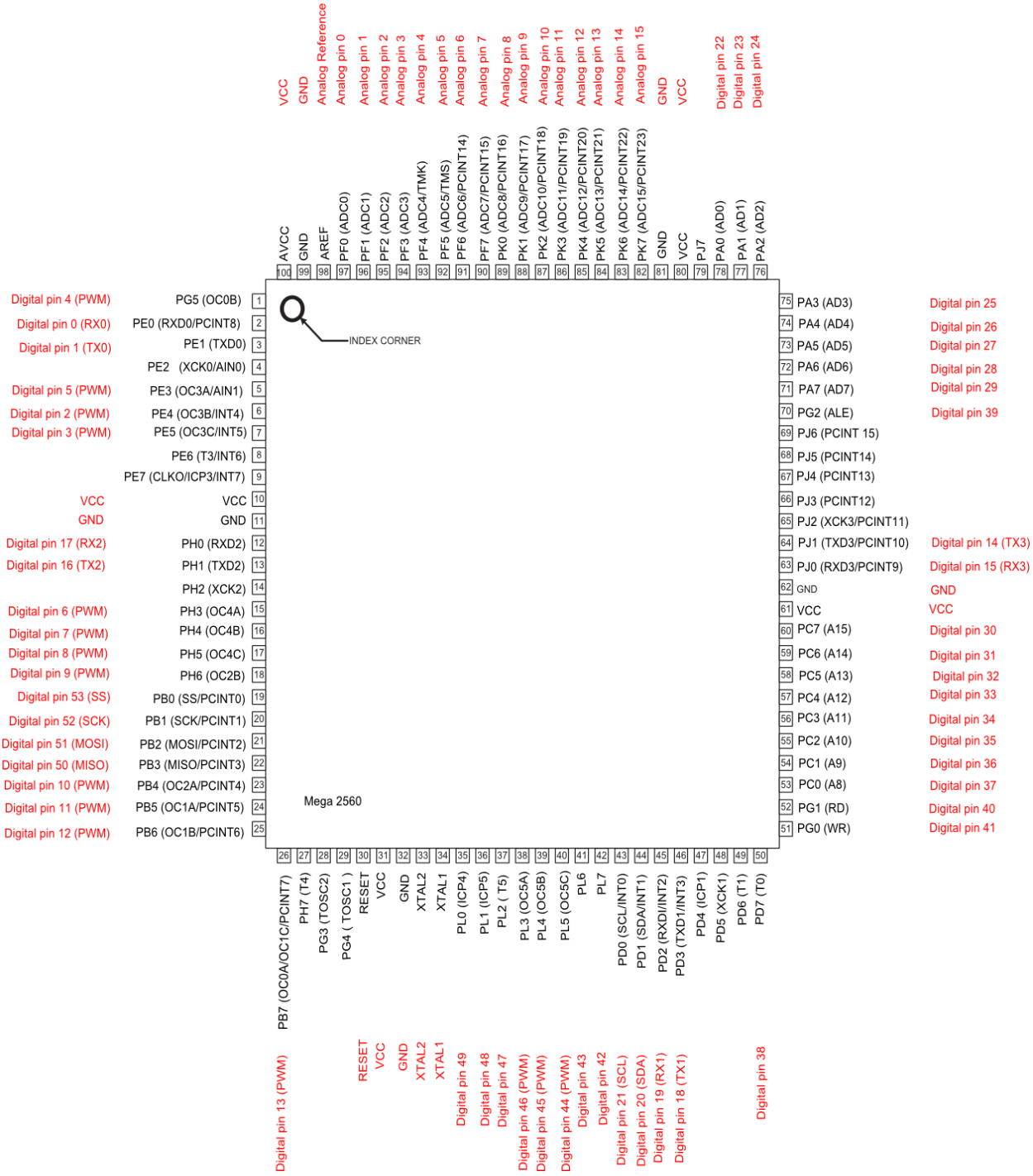


Figure 7: Figure: Arduino Mega 2560 PIN Diagram

Base Unit of Our Project:

Our objective was to make a miniature model of an autonomous car. The base unit of our project was an R/C car. Our aim was to turn this car into an autonomous vehicle guided by GPS. The model that we used in our project was a HSP monster truck.

Figure 8: Base Unit of Our Project

Features of the Vehicle:

- Four wheel drive system with shaft drive
- Fast Speed: 60km/hr
- Can run on any road surface
- Equipped with professional high performance brushless motor and bevel gear differential gear box
- Transmission shaft enables the car steadier, light shock fast speed while running
- High capacity 7.2V SC2000 mAh NI-MH rechargeable battery
- Front /rear bumper provide good protection against accident impacts
- Equipped with axletrees to reduce friction of the components and to run smoother and faster
- Front and read double wishbone suspension

Specifications of the Vehicle:

Length	400mm
Width	310mm
Height	185mm
Wheelbase	275mm
Gear Ratio	1:10.3
Weight	2267g
Wheel Diameter	120mm
Wheel Width	60mm
Engine	SP03302 Brushless Motor

Components of our Car:

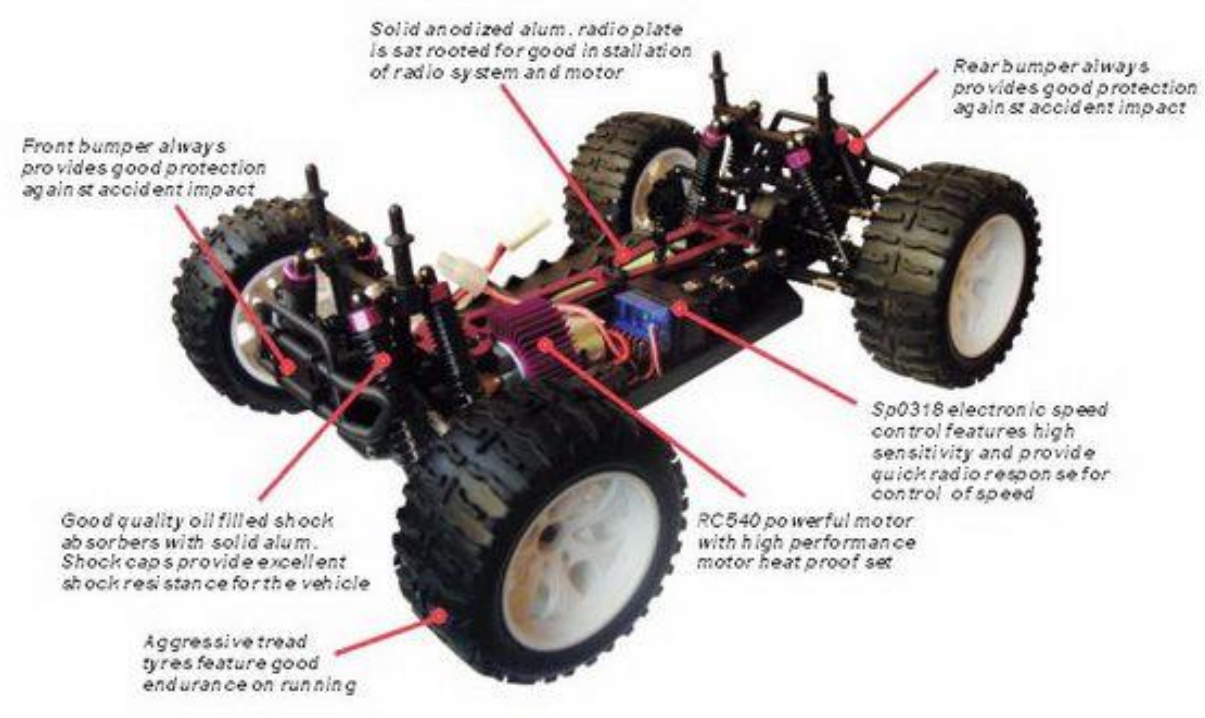


Figure 9: Different parts of the vehicle

The 4 wheeler robot vehicle includes different parts such as-

- speed control motor
- steering control motor
- battery
- tires
- suspension system etc.

Speed Controller:

- Controls speed of the vehicle
- Maximum speed up to 60km/hr

Figure 10: Electronic Speed Controller

Motor:

Figure 11: Speed control motor of the vehicle

Battery:

- 7.2 volt 1800mAh battery
- Works for up to half hour when fully charged

Figure 12: Battery

Heat guard:

- Absorbs heat emitted by the motor
- Made of aluminum for maximum heat absorption

Figure 13: Heat sink of the speed control motor

Suspension System:

Figure 14: : Suspension System of the vehicle

Tires:

- Wheelbase: 275mm
- Wheel diameter 120mm
- Wheel width 60mm

Figure 15: Tires

Assembly of the Vehicle:

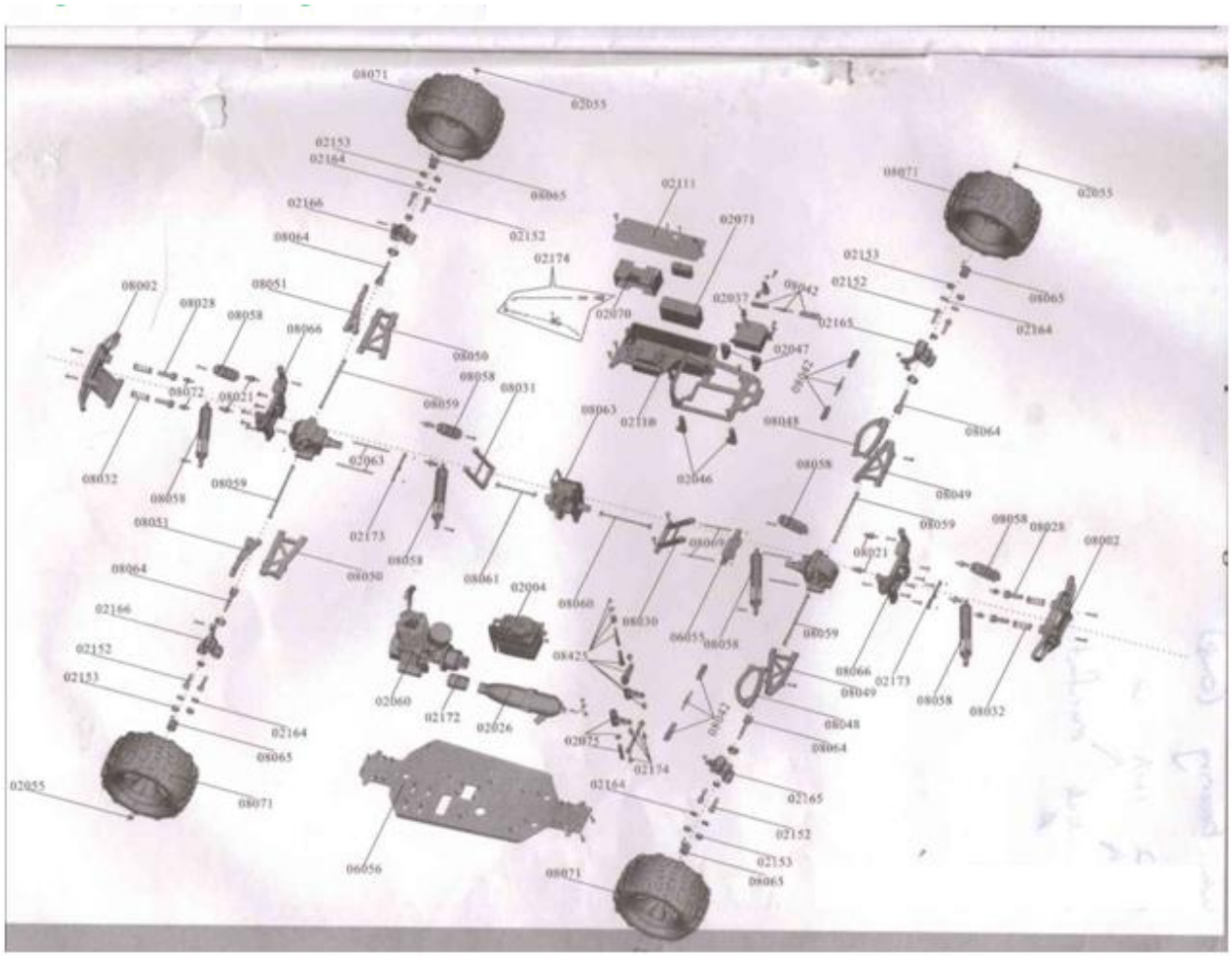


Figure 16: Assembly of the Vehicle

Communication:

Communication was done by two steps-

1. Communication between the car and the computer was done by using NRF.
2. Communication between the car and GPS satellite was done using Ublox GPS chip.

We gave GPS coordinates as input to the computer. GPS coordinates included the present location and the target location. This input was sent to the car with the help of NRF. The car read the inputs and moved according to the specified locations using the GPS chip.

RF Transceiver:

For the communication between our computer and the car we used an RF transceiver. It is a communication system which uses radio frequency to communicate. For the whole setup we needed a transmitter and a receiver. The transmitter was connected to the computer and the receiver was connected to the car. We needed two Arduino boards for the setup. One was used for the transmitter while the other was used for the receiver.

NRF24L01:

The RF transceiver that we used was the NRF24L01. It is a highly integrated, ultra low power RF transceiver IC for the 2.4GHz ISM band.

Specifications:

- Peak RX/TX currents lower than 14mA
- Sub μA power down mode
- Advanced power management
- 1.9 to 3.6V supply range
- Provides a true ULP solution

Figure 17:NRF24L01

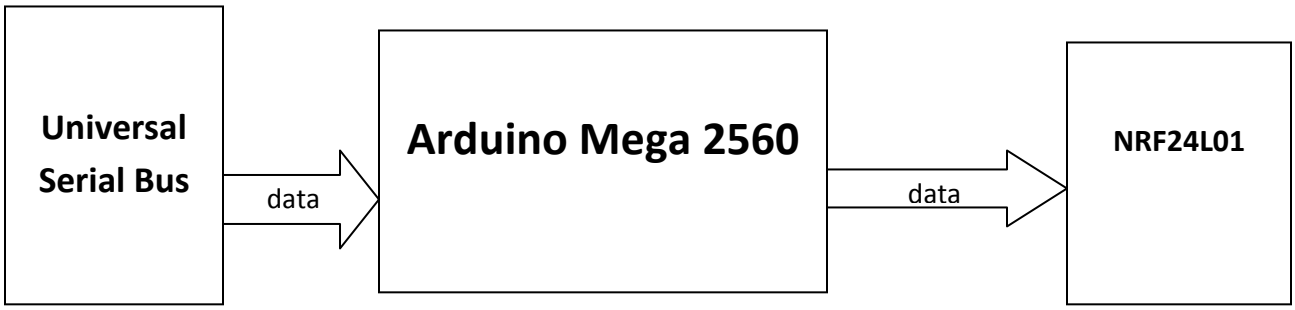
Setup of the RF Transceiver:

a) **Transmitter:**

For the transmitter we have used a total of 11 wires meaning 11 pins of the Arduino board. The pins used are as follows-

- 5 PWM pins
- RX1 pin
- 4 digital pins
- 3.3V power pin

The setup is shown below-



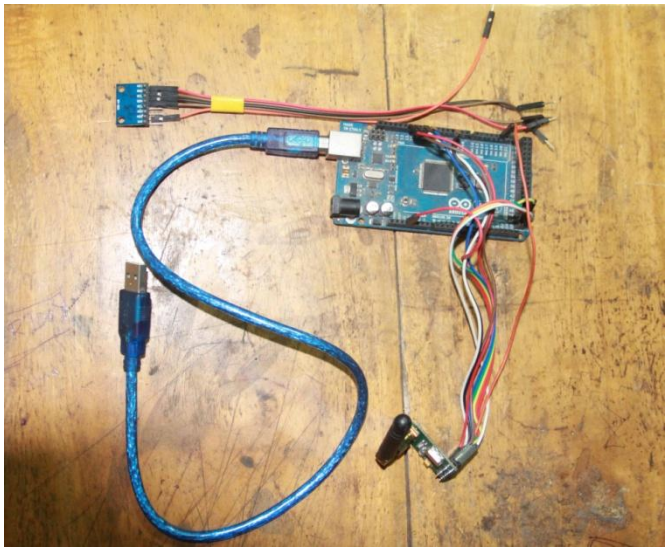
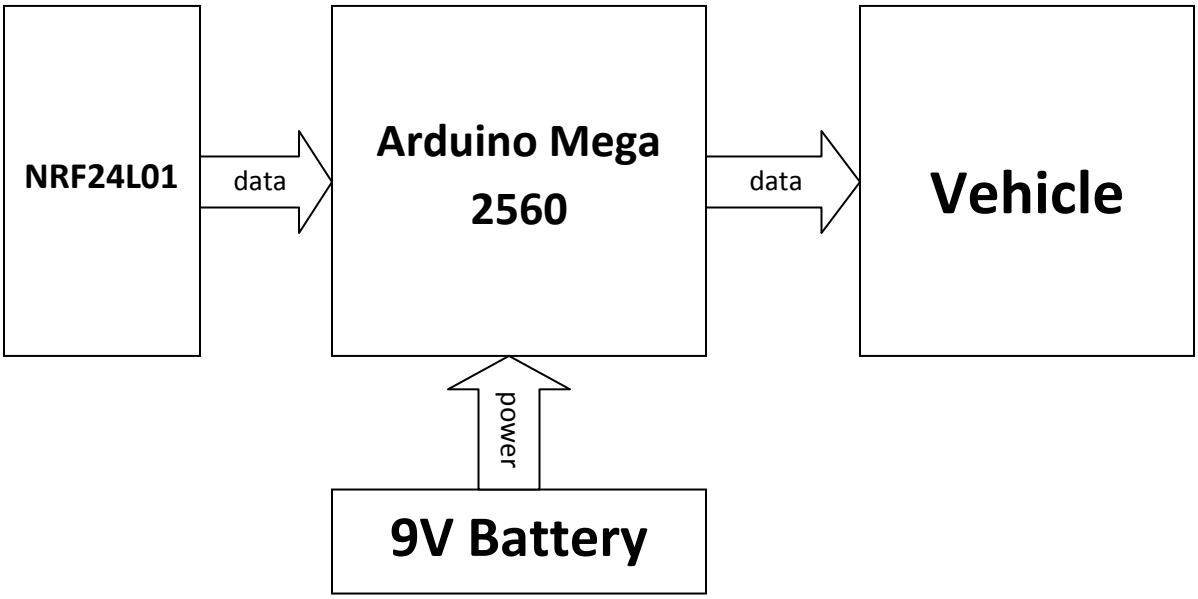


Figure 18: The Complete Setup of the Transmitter

b) Receiver:

For the receiver we used another similar Arduino board and another NRF24L01. The same 11 pins were used but this time the Arduino was set above the car, so the ESC, servo and the battery of the car were connected to the same Arduino board using an extra of 3 digital pins and a Vin pin.



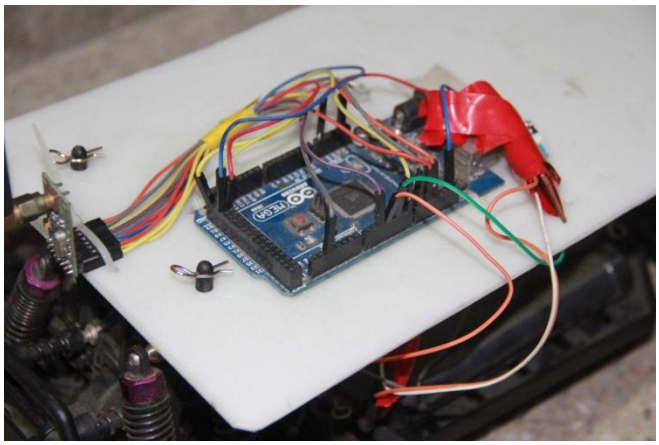


Figure 19: The Complete Setup of the Receiver

c) Making a Software for the NRF Communication:

We had to make a software for the transceiver to run. The software contained a main window from which we could control the car manually as well as setup the communication. By clicking the communication setup button we would enter another window. From this window we would select the port and the frequency.

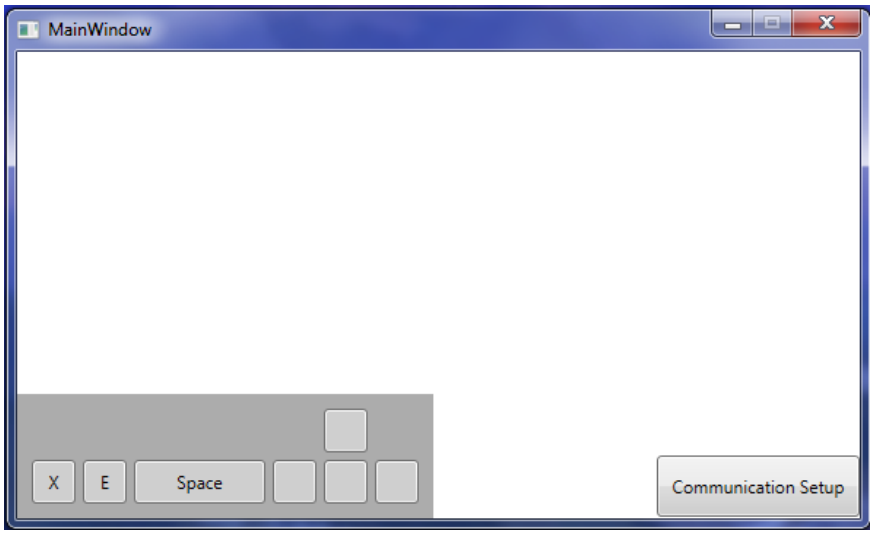


Figure 20: Main Window

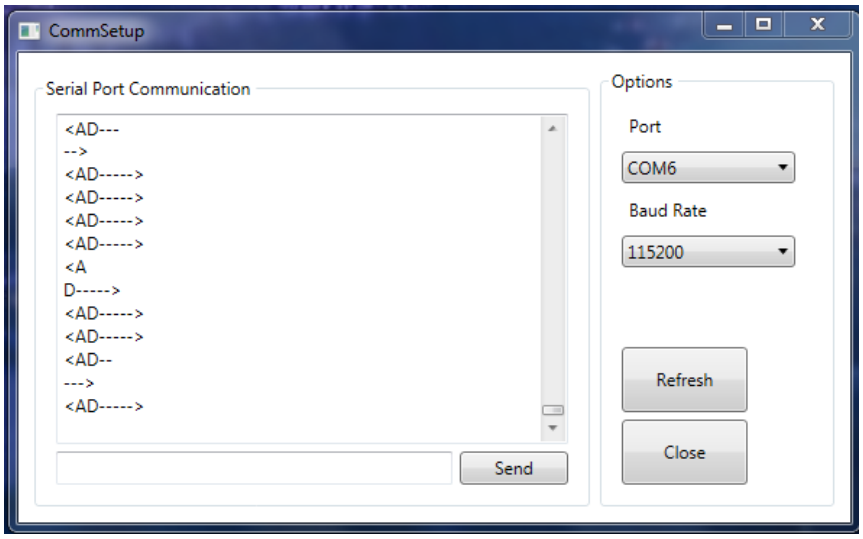
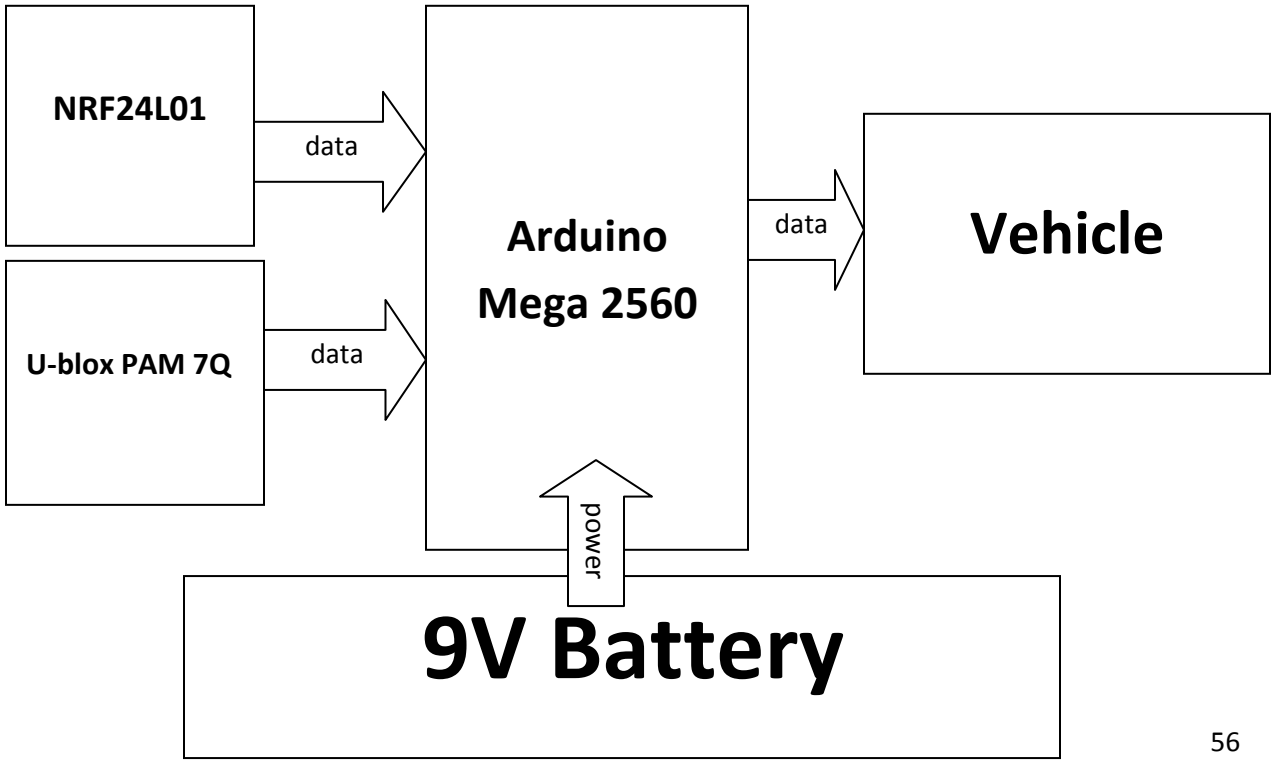


Figure 21: Command setup window

Setup of the GPS Chip:

We used a Ublox chip for the GPS communication of the car. We connected the GPS chip with the Arduino board on the car. This GPS chip was used to find the present location of the car as well as to identify the target location. The GPS chip was connected to the Arduino using 4 of its pins. The pins were Rx, Tx, Vin and Gnd.

Figure 22: U-blox PAM 7Q chip



Designing a Software for the GPS Chip:

We designed a software to give the input and the target locations. The software was a web browser based application. The inputs were given from a computer. The car communicated with the computer using the NRF communication described above.

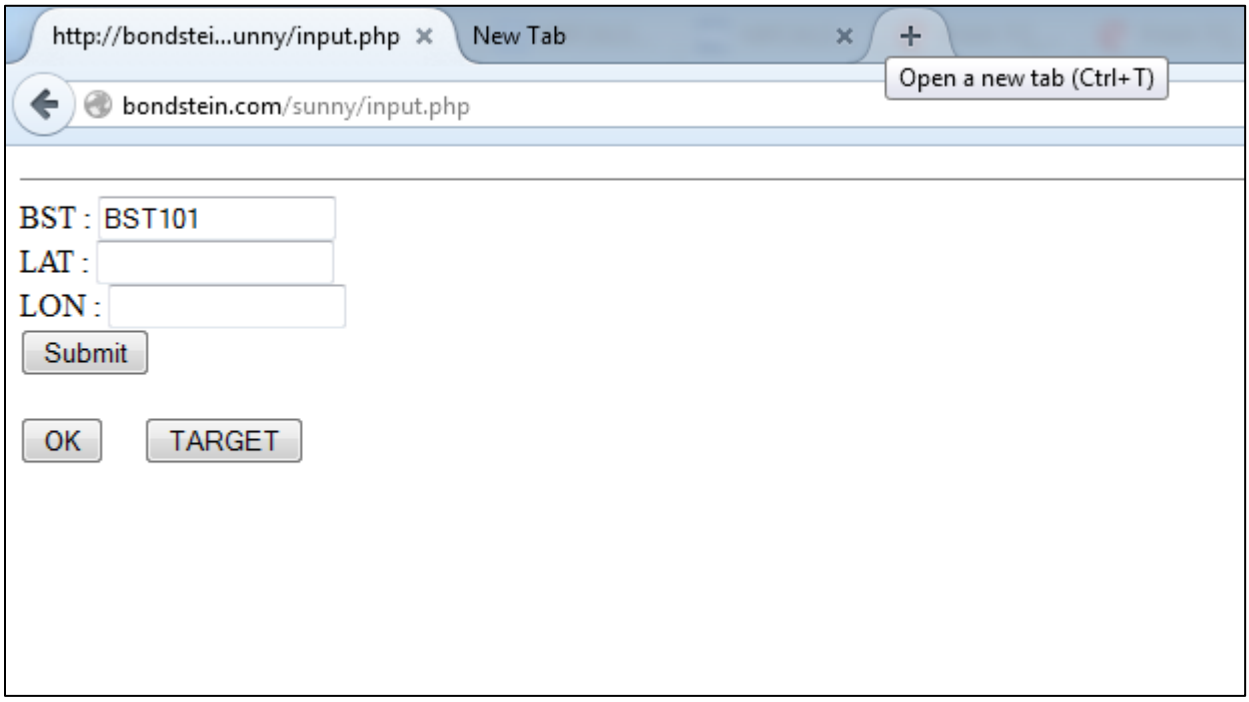


Figure 23: Web Application For the GPS Communication

Problems in Making the GPS work:

Unfortunately we couldn't make the GPS communication work. This may have happened due to the following reasons-

- There could be problem with the coding.
- The microprocessor couldn't handle the program calculations.
- The hardware setup could be incorrect.
- There could have been problem with the hardware.

- The software design might be suitable for the system.

With further time and experience we might have found the solution.

The Complete Setup of the Car:

The complete setup of the car includes the NRF receiver, the GPS chip, ESC, servo and the battery pack.

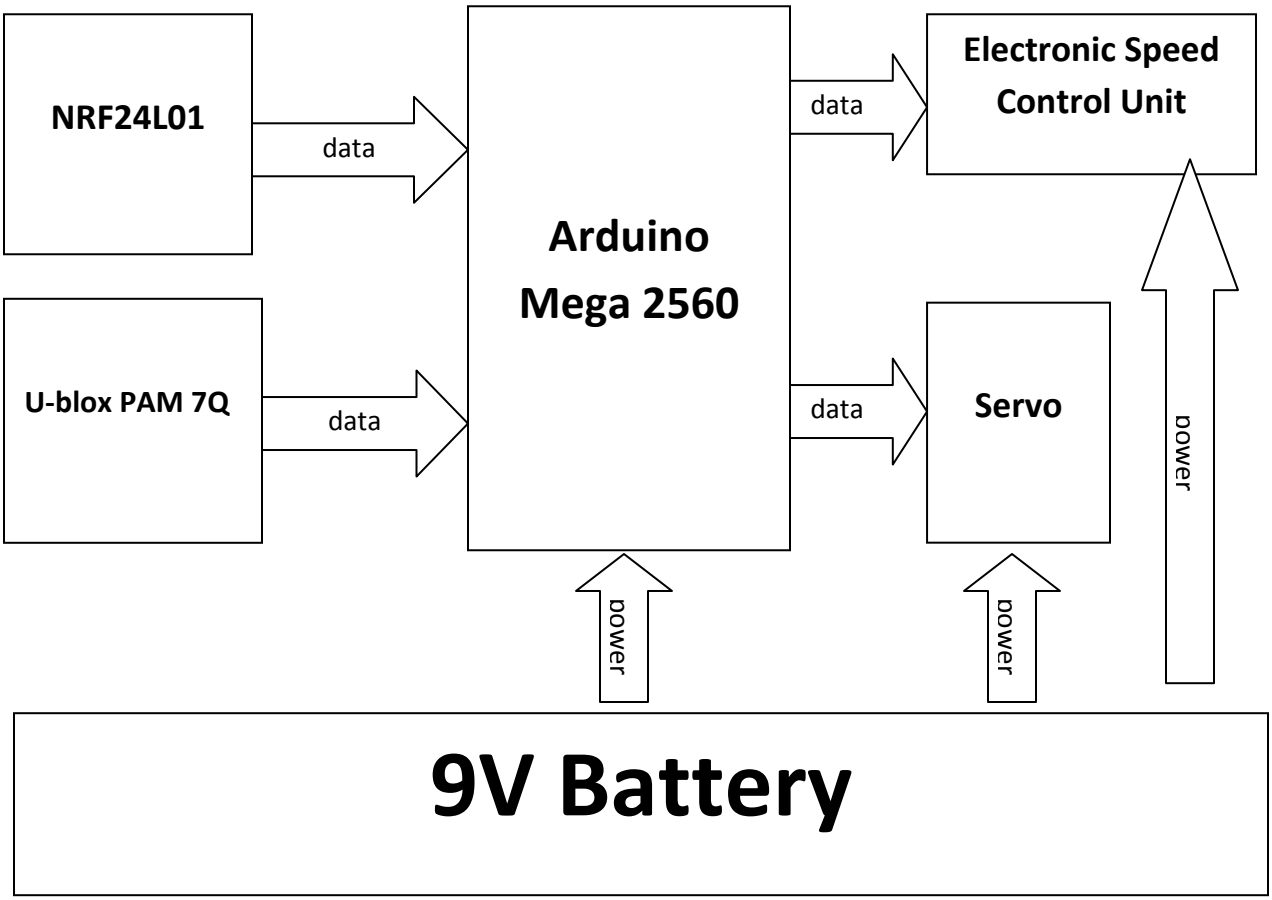
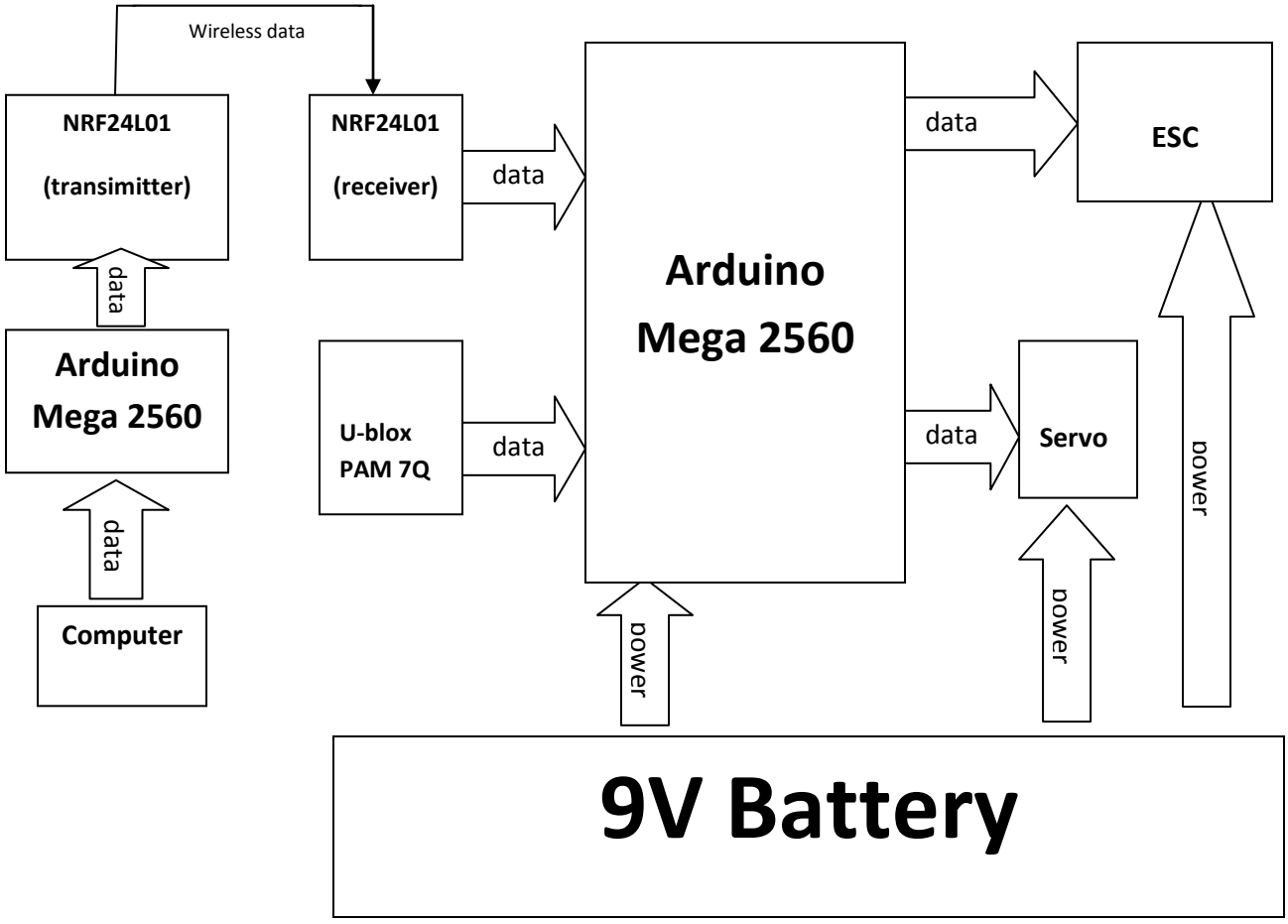


Figure 24: Complete Setup of the Car (without the GPS chip)

The Final Setup:

The final setup includes the complete car setup and the RF transmitter.



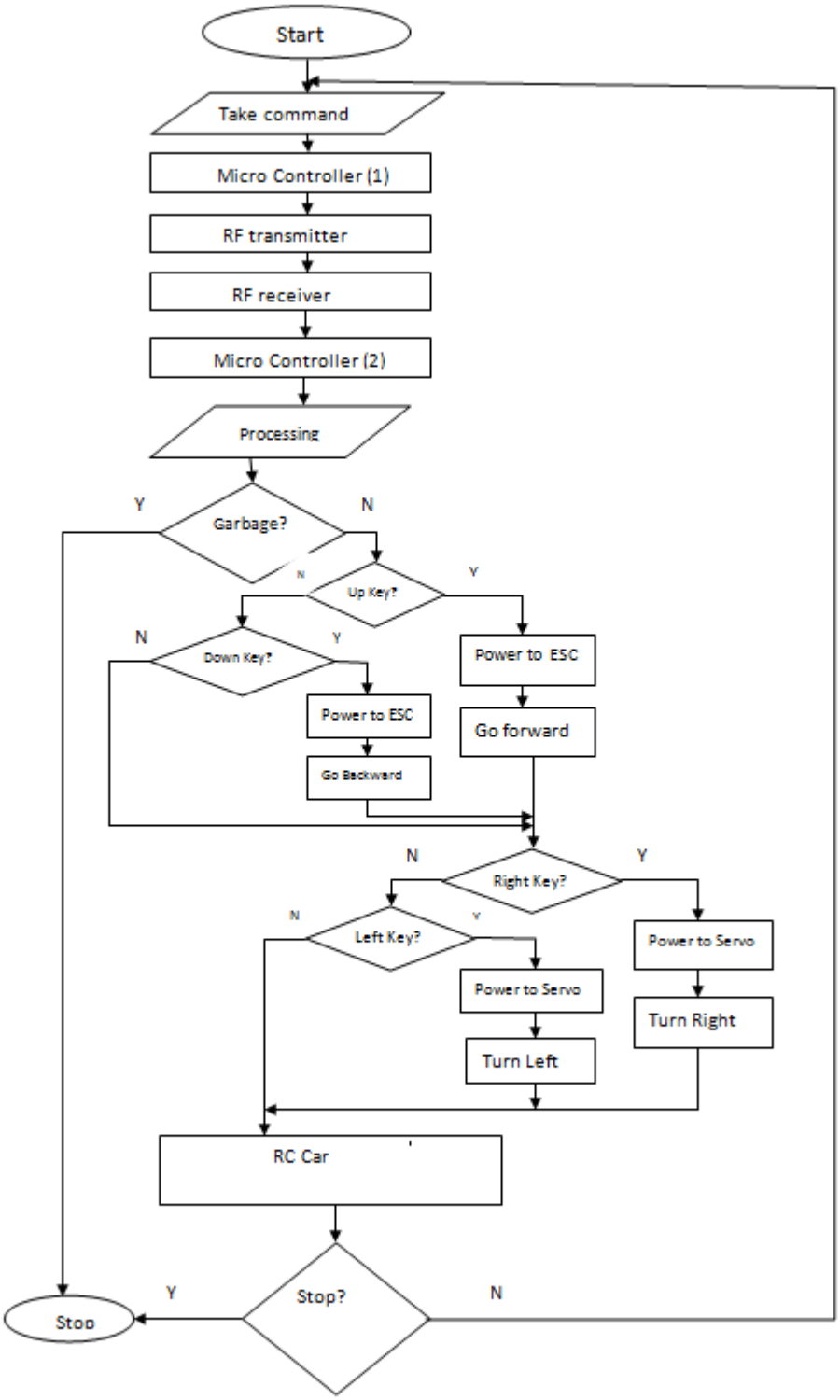
Program Schematic

Algorithm for the Communication between the Computer and the Car:

Step 1: start
Step 2: take command from user
Step 3: process command using micro-controller 1
Step 4: send data to RF transmitter
Step 5: receive data by RF receiver
Step 6: process data using micro-controller 2
Step 7: if data garbage then stop
Step 8: if data not garbage then proceed to step 9
Step 9: if up arrow key pressed then power to ESC and move forward
Step 10: if down key pressed then power to ESC and move backward
Step 11: if right arrow key pressed then power to servo and turn wheel right
Step 12: if left arrow key pressed then power to servo and turn wheel left
Step 13: if no key is pressed then stop and go to step 2
Step 14: stop

*Since we couldn't make the GPS work, we didn't give the algorithm for GPS communication.

Flow chart of the Program:



References:

[1] F. D. Boyden and S. A. Velinsky, "Limitations of kinematic models for wheeled mobile robots", Inter. Conf. On Advances in Robot Kinematics and Computationed Geometry, pp 252-260, 1994.

[2] M. Cherif, "Motion planning for all-terrain vehicles: A physical modeling approach for coping with dynamic and contact interaction constraints", IEEE Transactions on Robotics and Automation, Vol. 15, pp.202-218, 1999.

[3] J. R. Ellis, "Vehicle handling dynamics", Mechanical Engineering Publications, London, 1994.

[4] J. Fourquet and M. Renaud, "Time-optimal motions for a torque controlled wheeled mobile robot along specified paths", 35th Conf. On Decision and Control, pp. 3587-3592, Kobe, Japan, 1996.

[5] Th. Fraichard, "Dynamic trajectory planning with dynamic constraints: a 'State-time space' approach", IEEE Inter. Conf. On Intelligent Robots and Systems, pp.1391-1400, Yokohama, Japan, 1993.

[6] T. Fraichard and A.Scheuer, "Car-like Robots and moving obstacles", IEEE Inter. Conf. on Robotics and Automation, pp 64-69, the USA, 1994.

[7] F. Lamiraux, S. Sckhavat, and J. Laumond, "Motion planning and control for hilare pulling a trailer", IEEE Transactions on Robotics and Automation, Vol. 15, pp. 640-652, 1999.

[8] V. Munoz, A. Cruz and A. Garcia-Cerezo, "Speed planning and generation approach based on the path-time space for mobile robots", IEEE Inter. Conf. on Robotics and Automation, pp.2199-2204, Leuven, Belgium, 1998.

[9] Z. Shiller and Y.Gwo, "Dynamic motion planning of autonomous vehicles", IEEE Transactions on Robotics and Automation, Vol.7, pp.241-249, 1991.

[10] Z. Shiller and W. Serate, "Trajectory planning of tracked vehicles", J. of Dynamic Systems, Measurement, and Control, Vol. 117, pp. 619-624, 1995.

[11] W. Wu, H. Chen and P. Woo, "Optimal motion planning for a wheeled mobile robot", IEEE Inter. Conf. on Robotics and Automation, pp. 41-46, Detroit, Michigan, 1999.

[12] M. Yamamoto, M. Iwamura and A. Mohai, "Timeoptimal motion planning of skid-steer mobile robots in the presence of obastacles", IEEE Inter. Conf. on Robotics and Automation, pp. 32-37, Victoria, B.C., Canada, 1998.

[13] Trajectory Planning for a Four-Wheel-Steering Vehicle Danwei Wang Feng Qi School of Electrical and Electronic Engineering Nanyang Technological University

[14] http://www.bananahobby.com/hsp-1-10-brontosaurus-pro-off-road-rc-electric-powered-monster-truck.html

[15] Trajectory Planning in Robotics Alessandro Gasparetto· Paolo Boscariol·Albano Lanzutti· Renato Vidon

[16] Sciavicco, L., Siciliano, B., Villani, L., Oriolo, G.: Robotics. Modelling, Planning and Control. Springer, London (2009)

[17] Latombe, J.C.: Robot Motion Planning. Kluwer, Norwell (1991)

[18] Khatib,O.: Real-time obstacle avoidance for manipulators and mobile robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 500–505 (1985)

[19] Volpe, R.A, Khosla, P.K: Manipulator control with superquadric artificial potential functions: theory and experiments. IEEE Trans. Syst. Man Cybern.20(6), 1423–1436 (1990)

[20] Volpe, R.A.: Real and Artificial Forces in the Control of Manipulators: Theory and Experiments. Carnegie Mellon University,The Robotics Institute, Pittsburgh (1990)

[21] Koditschek, D.E.: Exact robot navigation using artificial potential functions. IEEE Trans. Robot. Autom.8(5), 501–518 (1992)

[22] Kim, J.O., Khosla, P.K.: Real-time obstacle avoidance using harmonic potential functions. IEEE Trans. Robot. Autom.8(3), 338–349 (1992)

[23] Connoly, C.I., Burns, J.B.: Path planning using Laplace's equation. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 2102–2106 (1990)

[24] Connoly, C.I., Grupen, RA.: On the application of harmonic functions to robotics. In: Proceedings of the IEEE International Symposium on Intelligent Control, pp. 498–502 (1992)

[25] Guldner, J., Utkin, V.I.: Sliding mode control for gradient tracking and robot navigation using artificial potential fields. IEEE Trans. Robot. Autom.11(2), 247–254 (1995) 278 A. Gasparetto et.al

[26] Ge, S.S., Cui, Y.J.: New potential functions for mobile robot path planning. IEEE Trans. Robot. Autom.16(5), 616–620 (2000)

[27] Barraquand, J., Latombe, J.C.: Robot motion planning: a distributed representation approach. Int. J. Robot. Res. 10(6), 628–649 (1991)

[28] Caselli, S., Reggiani, M., Sbravati, R.: Parallel path planning with multiple evasion strategies. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 260–266 (2002)

[29] Caselli, S., Reggiani M.: ERPP an experience-based randomized path planner. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1002–1008 (2000)

[20] Caselli, S., Reggiani, M., Rocchi R.: Heuristic methods for randomized path planning in potential fields. In: Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation, pp. 426–431 (2001)

[21] Amato, N.M., Wu, Y.: A randomized roadmap method for path and manipulation planning. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 113–120 (1996)

[22] Hsu, D., Kindel, R., Latombe, J.C., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. Int. J. Robot. Res.21(3), 233–255 (2002)

[23] Nissoux, C., Simon, T., Latombe, J.C.: Visibility based probabilistic roadmaps. In: Proceedings of the IEEE International Conference on Intelligent Robots and Systems, pp. 1316–1321 (1999)

[24] Clark, C.M., Rock S.: Randomized motion planning for groups of nonholomic robots. In: Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics and Automation in Space, pp. 1316–1321 (1999)

[25] Donald, B.R., Xavier, P.G.: Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open chain manipulators. In: Proceedings of the 6th Annual Symposium on Computational Geometry, pp. 290–300 (1990)

[26] Fraichard, T., Laugier, C.: Dynamic trajectory planning, path-velocity decomposition and adjacent paths. In: Proceedings of the 2nd International Joint Conference on Artificial Intelligence, pp. 1592–1597 (1993)

[27] Fiorini, P., Shiller, Z.: Time optimal trajectory planning in dynamic environments. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 1553–1558 (1996)

[28] Fraichard, T.: Trajectory planning in a dynamic workspace: a state-time space approach. Adv. Robot.13(1), 74–94 (1999)

[29] Kumar, V., Efran, M., Ostrowski, J.: Motion planning and control of robots. In: Handbook of Industrial Robotics. 2nd edn, Shimon, Y. Nof (ed) (1999)

[30] Gupta, K., del Pobil, A.P.: Practical Motion Planning in Robotics: Current Approaches and Future Directions. Wiley, West Sussex (1998)

[31] en.wikipedia.org/wiki/Microcontroller

[32] *www.arduino.cc/*