OBJECT RECOGNITION INTELLIGENT MOBILE ROBOT

Supervised By:

Dr. Abeer Twakol Khalil
Assistant Professor

Eng. Abdulrahman Elewah
Teaching Assistant

Prepared By:

Ibrahim Ahmed Taher
Ahmed Lotfy Badr
Ahmed Rafik Mohammed
Amin Mohammed Amin
Ahmed Safwat Abdelaziz
Kariem El-Sayed Khater

JULY 15, 2018
BENHA FACULTY OF ENGINEERING
Graduation Project

Acknowledgment

Any attempt at any level cannot be satisfactorily completed without the will of God and the Support and guidance of learned people. We would like to express our immense gratitude to *Prof. Abeer Tawakol* and *Eng. Abdulrahman Elewah* for their constant support and motivation that have encouraged us to come up with this project.

Abstract

Today's needs and efforts towards advancing technology for humanity is limitless. The way Machine Learning is changing how we perceive information, it has inspired solutions for a variety of everyday problems. With the advent of Machine Learning, objects will be capable of making their own decisions. Being an interesting area of research, there is a variety of techniques and algorithms for Machine Learning. Nonetheless we may not forget object recognition that is certainly one of the most exciting areas in machine learning right now. Computers have been able to recognize different objects even to the finest details like gestures and facial expressions, but recognizing arbitrary objects within a larger image has been the real deal. Surprisingly our brains effortlessly convert photons bouncing off objects at slightly different frequencies into a spectacularly rich set of information about the world around us. Machine learning still struggles with these simple tasks, but in the past few years, it's gotten much better.

Table of Content

Table of Figures

List of Tables

Introduction

The last few years have seen an explosion in the machine learning field, object recognition and artificial intelligence. The lowered cost of processors along with increasing the processing capabilities and getting introduced to new Nano-technologies has resulted in many products being made “smart” and able to make smart decisions.

Deep learning and a large public training data set called ImageNet has made an impressive amount of progress toward object recognition. A well-known framework that is used in deep learning algorithms’ implementation on a variety of architectures. It is very good at making use of GPUs, which themselves are great at implementing deep learning algorithms.

We are trying in our project to build a robot that could recognize objects. Days of research in building computer programs and doing test-driven development have turned us into a menace working on physical projects. In our real world, testing your device for bugs can set your lap on fire, or at least burn up your components.

CHAPTER 1:

Introduction to Artificial Intelligence

Learning field, object recognition and artificial intelligence. The lowered cost of processors along with increasing the processing capabilities and getting introduced to new Nano-technologies has resulted in many products being made “smart” and able to make smart decisions.

Deep learning and a large public training data set called ImageNet has made an impressive amount of progress toward object recognition. A well-known framework that is used in deep learning algorithms’ implementation on a variety of architectures. It is very good at making use of GPUs, which themselves are great at implementing deep learning algorithms.

We are trying in our project to build a robot that could recognize objects. Days of research in building computer programs and doing test-driven development have turned us into a menace working on physical projects. In our real world, testing your device for bugs can set your lap on fire, or at least burn up your components.

1.1 Artificial Intelligence (AI)

Artificial Intelligence (AI) is everywhere. Possibility is that you are using it in one way or the other and you don’t even know about it. One of the popular applications of AI is Machine Learning (ML), in which computers, software, and devices perform via cognition (very similar to human brain). Herein, we share few examples of machine learning that we use every day and perhaps have no idea that they are driven by ML.

So we can say the AI is what people call computer programs that try to replicate how the human brain operates. For now, they only can replicate very specific tasks where Machine mimics cognitive functions so in computer science AI research is defined as the study of "intelligent agents" These are any devices that perceive its environment and takes actions that maximize its chance of success at some goal.

1.1.1 Artificial Intelligence Goals

- Goal 1: Measure our progress

A significant fraction of our research bandwidth is being spent on fundamental research. We'll always be developing and testing new ideas, especially those that don't fit neatly into our current worldview. This is important — our current ideas will not be enough to achieve our long-term goal.

We've also formed teams around specific projects. The intention isn't just to solve these problems, but to develop general learning algorithms in the process. These algorithms will, in turn, help us build agents that are more capable according to our metric. These projects are:

- Goal 2: Build a household robot

We're working to enable a physical robot (off-the-shelf; not manufactured by OpenAI) to perform basic housework. There are existing techniques for specific tasks, but we believe that learning algorithms can eventually be made reliable enough to create a general-purpose robot. More generally, robotics is a good testbed for many challenges in AI.

- Goal 3: Build an agent with useful natural language understanding

We plan to build an agent that can perform a complex task specified by language, and ask for clarification about the task if it's ambiguous. Today, there are promising algorithms for supervised language tasks such
as question answering, syntactic parsing and machine translation but there aren't any for more advanced linguistic goals, such as the ability to carry a conversation, the ability to fully understand a document, and the ability to follow complex instructions in natural language. We expect to develop new learning algorithms and paradigms to tackle these problems.

- Goal 4: Solve a wide variety of games using a single agent

We aim to train an agent capable enough to solve any game in our initial metric. Games are virtual mini-worlds that are very diverse, and learning to play games quickly and well will require significant advances in generative
models and reinforcement learning. (We are inspired by the pioneering work of DeepMind, who have produced impressive results in this area in the past few years.)

Our projects and fundamental research all have shared cores, so progress on any is likely to benefit the others. Each captures a different aspect of goal-solving, and was chosen for its potential to significantly move our metric.

We're just getting started on these projects, and the details may change as we gain additional data. We also expect to add new projects over time.

If you're excited about any of the above, we'd love to hear from you, whether by discussing with others in the Open AI community or joining us full-time.

So in general we can say that the (AI) seeks to:

- Make robots think on human neuron level
- Have robots that are capable of logical thinking
- Allow intelligent agents to understand different languages and communicate
- Create agents able to analyze data and make decisions based on processed information

1.1.2 Artificial Intelligence Applications

Samuel's checker player

In 1959, after the first commercial computers became available, Arthur Lee Samuel, an early researcher in artificial intelligence, developed a computer program to play checkers. His program had a scoring function based on the position of the board at any given time and had various mechanisms by which the program could learn and become better, such as reevaluating the scoring function based on input from professional games and by playing thousands of games against itself. The program achieved sufficient skill to challenge good players.

Natural Language Processing (NLP)

Handling human languages has been one of the most sought-after goals of artificial intelligence. The nuances and ambiguities of language pose formidable problems for researchers. Idioms like "cut the mustard" and "beat around the bush" cannot always be translated into other languages without a lot of explanation.

Linguistic analysis generally relies on a lexicon (dictionary) where the words contain syntactic and semantic attributes that can be used for disambiguation. Morphological

analysis is used to obtain the roots and suffixes of the words. Syntactic analysis or grammar determines the relationship between the words and identifies well-structured grammatical units. Semantic analysis puts together the syntactic units to deduce the meaning of the text. For text searching applications, the semantic units may be supplemented with synonyms so that a query for "bird flu" can retrieve documents about "avian influenza".

Natural language is the fundamental way in which humans will interface with AI devices. Advances in NLP will benefit AI and automatic translation, which is very important for global marketing by multinational organizations.

Manipulation and Control of Objects

A computer with artificial intelligence could take a spoken or written question and provide the answer on a screen. However, we expect AI computers to do more than that. We want our computers to turn on the coffee pot in the morning, regulate the temperature in our house and warm up the car when we are ready to go to work.

Robots are in fact mobile AI computers. We could say that robots include airplanes with autopilots that can control takeoff, navigation and landing. Self-driving cars such as those developed by Google can also be considered AI robots. Robot technology generally incorporates collision avoidance to prevent accidents.

Turing test

The Turing test determines a machine's ability to exhibit intelligent behavior equivalent or indistinguishable from that of a human. In 1950, Alan Turing proposed a test consisting of a human evaluator, a machine designed to generate human-like responses, and a human participant. All three would be separated from one another, and the evaluator would try to determine which of the other two was the machine by asking questions using a computer keyboard. If the evaluator cannot reliably tell the machine from the human, the machine would pass the test.

AI Weapons

SpaceX founder Elon Musk, physicist Stephen Hawking, Apple co-founder Steve Wozniak and other prominent people in technology, robotics, and artificial intelligence have warned against the military use of AI weapons because this could start a global arms race to produce autonomous killing machines. Smart bombs with the appearance of ordinary household items, but fitted with speech recognition and face recognition could be programmed to detonate only when a particular individual is in close proximity. Targeted killings of terrorist leaders could be the first applications of such AI weapons

Siri

Everyone is familiar with Apple's personal assistant, Siri. She's the friendly voice-activated computer that we interact with on a daily basis. She helps us find information, gives us directions, add events to our calendars, helps us send messages and so on. Siri is a pseudo-intelligent digital personal assistant. She uses machine-learning technology to get smarter and better able to predict and understand our natural-language questions and requests.

Tesla

If you don't own a Tesla, you have no idea what you're missing. This is quite possibly one of the best cars ever made. Not only for the fact that it's received so many accolades, but because of its predictive capabilities, self-driving features and sheer technological "coolness." Anyone that's into technology and cars needs to own a Tesla, and these vehicles are only getting smarter and smarter thanks to their over-the-air updates

1.1.3 Evolution of Artificial Intelligence

The goal of creating non-biological intelligence has been with us for a long time, predating the nominal 1956 establishment of the field of artificial intelligence by centuries or, under some definitions, even by millennia. For much of this history it was reasonable to recast the goal of “creating” intelligence as that of “designing” intelligence. For example, it would have been reasonable in the 17th century, as Leibnitz was writing about reasoning as a form of calculation, to think that the process of creating artificial intelligence would have to be something like the process of creating a waterwheel or a pocket watch: first understand the principles, then use human intelligence to devise a design based on the principles, and finally build a

system in accordance with the design. At the dawn of the 19th century William Paley made such assumptions explicit, arguing that intelligent designers are necessary for the production of complex adaptive systems. And then, of course, Paley was soundly refuted by Charles Darwin in 1859. Darwin showed how complex and adaptive systems can arise naturally from a process of selection acting on random variation. That is, he showed that complex and adaptive design could be created without an intelligent designer. On the basis of evidence from paleontology, molecular biology, and evolutionary theory we now understand that nearly all of the interesting features of biological agents, including intelligence, have arisen through roughly Darwinian evolutionary processes (with a few important refinements, some of which are mentioned below). But there are still some holdouts for the pre-Darwinian view. A recent survey in the United States found that 42% of respondents expressed a belief that "Life on Earth has existed in its present form since the beginning of time", and these views are supported by powerful political forces including a stridently anti-science President. These shocking political realities are, however, beyond the scope of the present essay. This essay addresses a more subtle form of pre-Darwinian thinking that occurs even among the scientifically literate, and indeed even among highly trained scientists conducting advanced AI research. Those who engage in this form of pre-Darwinian thinking accept the evidence for the evolution of terrestrial life but ignore or even explicitly deny the power of evolutionary processes to produce adaptive complexity in other contexts. Within the artificial intelligence research community those who engage in this form of thinking ignore or deny the power of evolutionary processes to create machine intelligence. Before exploring this complaint further it is worth asking whether an evolved artificial intelligence would even serve the broader goals of AI as a field. Every AI text opens by defining the field, and some of the proffered definitions are explicitly oriented toward design—presumably design by intelligent humans. For example Dean et al. define AI as "the design and study of computer programs that behave intelligently". Would the field, so defined, be served by the demonstration of an evolved artificial intelligence? It would insofar as we could study the evolved system and particularly if we could use our resulting understanding as the basis for future designs. So even the most design-oriented AI researchers should be interested in evolved artificial intelligence if it can in fact be created.

Most AI texts nonetheless demonstrate, by their coverage, that their authors view AI as a set of design problems that human designers are expected to solve. Few describe evolutionary perspectives on the development of natural or artificial intelligence, and

most describe existing evolutionary methods (usually just genetic algorithms) only as an odd form of search or learning.1 But these same texts are generally less committed to human design in their introductory definitions. For example Winston states that AI is "the study of ideas that enable computers to be intelligent", while Charniak and McDermott say that AI is "the study of mental faculties through the use of computational methods". Neither of these definitions seems particularly wedded to design. If they, and others like them, truly capture the over-arching objectives of the field then it would seem that evolved intelligence—the only kind of general intelligence of which we currently have examples—should be of central interest, as should be the conditions under which intelligence can evolve. So AI researchers should be interested in evolved artificial intelligence in principle, but is there any reason to believe that evolution can produce artificial intelligence in practice? Traditional genetic algorithms have not done so, at least not in any general sense, but evolutionary computation is rapidly advancing, fueled in part by revolutions in our understanding of biological evolution. Darwin, of course, didn't have the whole story on biological evolution, and neither did the developers of the first genetic algorithms. Darwin was ignorant not only of the relevant molecular biology but also of population and gene dynamics and of many other mechanisms and principles that are central to a modern understanding of evolution. We now know, for example, that biological evolution depends in important ways on genetic representation, on ecological interactions within and among populations, and on genetic control of gene expression, reproductive strategies, development, and learning. None of these features of biological evolution were incorporated into the first evolutionary computation systems—ideas for which, incidentally, date at least to Turing but serious work has been devoted to several of them in recent years. The problem-solving performance of evolutionary algorithms has advanced significantly in the past decade or so, to the extent that human-competitive results have recently been achieved in several areas of science and engineering; these include evolved designs for antennas, photonic crystals, quantum computer algorithms, and even search heuristics. Many of these results have been achieved using systems that incorporate at least a few of the insights from recent biological advances for example, many involve the genetic representation of developmental processes instead of "adult" phenotypes but there is still a long way to go if the computational work is to catch up with biology. Nonetheless, work in evolutionary computation is moving rapidly forward and it is doing so within an increasingly mature and stable research community. One indication of this progress is the recent establishment, by the Association for Computing Machinery, of a full-fledged Special Interest Group on Genetic and Evolutionary Computation (SIGEVO). None of this means that we should expect a

simple extension of current evolutionary algorithms to produce general artificial intelligence any time soon. This is in part because the field of evolutionary computation has proceeded largely in isolation from developments in mainstream AI that will surely undergird future intelligent systems. But the shortest path to achieving AI's long-term goals will probably involve both human design and evolution. It is not yet clear what sorts of interactions between design and evolution will prove to be most helpful or from which research communities they will emerge. It is possible that the long-established tradition of analyzing evolution as a form of search will continue to flower to the extent that it drives a broader integration of evolutionary thinking and mainstream AI. Or perhaps the integration will be driven by successful evolutionary approaches to mainstream AI problems such as RoboCup. Or maybe the integration will grow out of work in machine learning, an area that has traditionally provided formal models of evolution and which may also provide mechanisms to manage adaptation over multiple time scales (encompassing processes normally described as learning, development, and evolutionary change).

Whenever and wherever such integration occurs it may take a variety of forms. For example, future developers might use representations and algorithms that have been developed in mainstream AI as the ingredients in the "primordial ooze" that is sampled and recombined by evolution. Alternatively, they might use human-designed algorithms to intelligently compose components that have been devised and refined by evolutionary methods. Or they may combine design and evolution in ways that nobody has yet imagined. In any event the prevailing assumption that general artificial intelligence can only be (or should only be) designed by intelligent human designers is flawed and should be rejected. That this pre-Darwinian assumption is indeed prevalent is demonstrated not only by the lack of evolutionary perspectives in AI texts but also by the niche treatment of evolution as a special purpose search or learning algorithm in most of the mainstream AI research literature. A more appropriate post-Darwinian research strategy would consider evolutionary methods for any problem domain in which human design is difficult, and would expect evolutionary methods to be increasingly useful in increasingly general problem environments. Even when intelligent human design is appropriate, progress toward the field's over-arching goals would be served by situating the work within a larger evolutionary context. The field has long acknowledged that intelligent behavior might best be achieved by agents that teach agents that grow or redesign themselves to some limited extent as they confront their environments. But modern biology and technological advances both support a more radical offloading of design from humans to the intelligent systems themselves and to their ecologies. Not only should

the AI systems of the future grow and learn, but their developmental and learning processes should be crafted by the most powerful designer of adaptive complexity known to science: natural selection. In some senses AI has been moving in this direction since 1956. The field's historical trajectory from a focus on isolated and sophisticated mental faculties to a focus on the commonsense knowledge needed for everyday tasks, and more recently to a focus on the construction of complete, situated, and embodied agents, is a trajectory from a priori conceptions of intelligence toward theories derived from the natural forms of intelligence that we observe around us. The logical extension of this trend is to model not only the products of natural evolution but also its processes.

1.2 Machine Learning

What is machine learning?

Machine learning is the idea that there are generic algorithms that can tell you something interesting about a set of data without you having to write any custom code specific to the problem.

1.2.1 Machine learning applications

i. Image Recognition:

One of the most common uses of machine learning is image recognition. There are many situations where you can classify the object as a digital image. For digital images, the measurements describe the outputs of each pixel in the image.

In the case of a black and white image, the intensity of each pixel serves as one measurement. So if a black and white image has N*N pixels, the total number of pixels and hence measurement is N2.

In the colored image, each pixel considered as providing 3 measurements to the intensities of 3 main color components ie RGB. So N*N colored image there are 3 N2 measurements.

For face detection – The categories might be face versus no face present. There might be a separate category for each person in a database of several individuals.

For character recognition – We can segment a piece of writing into smaller images, each containing a single character. The categories might consist of the 26 letters of the English alphabet, the 10 digits, and some special characters.

ii. Speech Recognition

Speech recognition (SR) is the translation of spoken words into text. It is also known as "automatic speech recognition" (ASR), "computer speech recognition", or "speech to text" (STT).

In speech recognition, a software application recognizes spoken words. The measurements in this application might be a set of numbers that represent the speech signal. We can segment the signal into portions that contain distinct words or phonemes. In each segment, we can represent the speech signal by the intensities or energy in different time-frequency bands.

Although the details of signal representation are outside the scope of this program, we can represent the signal by a set of real values.

Speech recognition applications include voice user interfaces. Voice user interfaces are such as voice dialing, call routing, domestic appliance control. It can also use as simple data entry, preparation of structured documents, speech-to-text processing, and plane.

iii. Medical Diagnosis

ML provides methods, techniques, and tools that can help solving diagnostic and prognostic problems in a variety of medical domains. It is being used for the analysis of the importance of clinical parameters and of their combinations for prognosis, e.g. prediction of disease progression, for the extraction of medical knowledge for outcomes research, for therapy planning and support, and for overall patient management. ML is also being used for data analysis, such as detection of regularities in the data by appropriately dealing with imperfect data, interpretation of continuous data used in the Intensive Care Unit, and for intelligent alarming resulting in effective and efficient monitoring.

It is argued that the successful implementation of ML methods can help the integration of computer-based systems in the healthcare environment providing opportunities to facilitate and enhance the work of medical experts and ultimately to improve the efficiency and quality of medical care.

In medical diagnosis, the main interest is in establishing the existence of a disease followed by its accurate identification. There is a separate category for each disease under consideration and one category for cases where no disease is present. Here, machine learning improves the accuracy of medical diagnosis by analyzing data of patients.

The measurements in this application are typically the results of certain medical tests (example blood pressure, temperature and various blood tests) or medical diagnostics (such as medical images), presence/absence/intensity of various symptoms and basic physical information about the patient (age, sex, weight ..etc.). On the basis of the results of these measurements, the doctors narrow down on the disease inflicting the patient.

iv. Statistical Arbitrage

n finance, statistical arbitrage refers to automated trading strategies that are typical of a short term and involve a large number of securities. In such strategies, the user tries to implement a trading algorithm for a set of securities on the basis of quantities such as historical correlations and general economic variables. These measurements can be cast as a classification or estimation problem. The basic assumption is that prices will move towards a historical average.

We apply machine learning methods to obtain an index arbitrage strategy. In particular, we employ linear regression and support vector regression (SVR) onto the prices of an exchange-traded fund and a stream of stocks. By using principal component analysis (PCA) in reducing the dimension of feature space, we observe the benefit and note the issues in the application of SVR. To generate trading signals, we model the residuals from the previous regression as a mean reverting process.

In the case of classification, the categories might be sold, buy or do nothing for each security. I the case of estimation one might try to predict the expected return of each security over a future time horizon. In this case, one typically needs to use the estimates of the expected return to make a trading decision (buy, sell, etc.)

v. Learning Associations

Learning association is the process of developing insights into various associations between products. A good example is how seemingly unrelated products may reveal

an association to one another. When analyzed in relation to buying behaviors of customers.

One application of machine learning- Often studying the association between the products people buy, which is also known as basket analysis. If a buyer buys 'X', would he or she force to buy 'Y' because of a relationship that can identify between them. This leads to relationship that exists between fish and chips etc. when new products launches in the market a Knowing these relationships it develops new relationship. Knowing these relationships could help in suggesting the associated product to the customer. For a higher likelihood of the customer buying it, It can also help in bundling products for a better package.

This learning of associations between products by a machine is learning associations. Once we found an association by examining a large amount of sales data, Big Data analysts. It can develop a rule to derive a probability test in learning a conditional probability.

vi. Classification

A Classification is a process of placing each individual from the population under study in many classes. This is identify as independent variables.

Classification helps analysts to use measurements of an object to identify the category to which that object belongs. To establish an efficient rule, analysts use data. Data consists of many examples of objects with their correct classification.

For example, before a bank decides to disburse a loan, it assesses customers on their ability to repay the loan. By considering factors such as customer's earning, age, savings and financial history we can do it. This information taken from the past data of the loan. Hence, Seeker uses to create relationship between customer attributes and related risks.

vii. Prediction

Consider the example of a bank computing the probability of any of loan applicants faulting the loan repayment. To compute the probability of the fault, the system will

first need to classify the available data in certain groups. It is described by a set of rules prescribed by the analysts.

Once we do the classification, as per need we can compute the probability. These probability computations can compute across all sectors for varied purposes

Current prediction is one of the hottest machine learning algorithms. Let's take an example of retail, earlier we were able to get insights like sales report last month / year / 5-years / Diwali / Christmas. These type of reporting is called as historical reporting. But currently business is more interested in finding out what will be my sales next month / year / Diwali, etc.

So that business can take required decision (related to procurement, stocks, etc.) on time.

vii. Extraction

Information Extraction (IE) is another application of machine learning. It is the process of extracting structured information from unstructured data. For example web pages, articles, blogs, business reports, and e-mails. The relational database maintains the output produced by the information extraction.

The process of extraction takes input as a set of documents and produces a structured data. This output is in summarized form such as excel sheet and table in a relational database.

Now-a-days extraction is becoming a key in big data industry.

As we know that huge volume of data is getting generated out of which most of the data is unstructured. The first key challenge is handling of unstructured data. Now conversion of unstructured data to structured form based on some pattern so that the same can stored in RDBMS.

Apart from this in current days data collection mechanism is also getting change. Earlier we collected data in batches like End-of-Day (EOD), but now business want the data as soon as it is getting generated, i.e. in real time.

viii. Regression

We can apply Machine learning to regression as well.

Assume that x= x1, x2, x3 … xn are the input variables and y is the outcome variable. In this case, we can use machine learning technology to produce the output (y) on the basis of the input variables (x). You can use a model to express the relationship between various parameters as below:

Y = g(x) where g is a function that depends on specific characteristics of the model.

In regression, we can use the principle of machine learning to optimize the parameters. To cut the approximation error and calculate the closest possible outcome.

We can also use Machine learning for function optimization. We can choose to alter the inputs to get a better model. This gives a new and improved model to work with. This is known as response surface design.

Conclusion

In conclusion, Machine learning is an incredible breakthrough in the field of artificial intelligence. While it does have some frightening implications when you think about it, these Machine Learning Applications are several of the many ways this technology can improve our lives.

1.3 Hot AI Technologies

There are many different technologies depending on AI such as:

i. Natural Language Generation

ii. Speech Recognition

iii. Computer Vision

v. Virtual Agents

vi .AI-Optimized Hardware

vii. Decision Management

viii. Deep Learning Platforms

ix. Biometrics

x. Robotic Process Automation

xi. Text Analytics and NLP

1.3.1 Computer Vision

Assuming a ball is thrown to you and you want to catch it...
Actually, this is one of the most complex processes we've ever attempted to comprehend – let alone recreate. Inventing a machine that sees like we do is a deceptively difficult task, not just because it's hard to make computers do it, but because we're not entirely sure how we do it in the first place.

What actually happens is roughly this: the image of the ball passes through your eye and strikes your retina, which does some elementary analysis and sends it along to the brain, where the visual cortex more thoroughly analyzes the image. It then sends it out to the rest of the cortex, which compares it to everything it already knows, classifies the objects and dimensions, and finally decides on something to do: raise your hand and catch the ball (having predicted its path). This takes place in a tiny fraction of a second, with almost no conscious effort, and almost never fails. So recreating human vision isn't just a hard problem, it's a set of them, each of which relies on the other.

Well, no one ever said this would be easy. Except, perhaps, AI pioneer Marvin Minsky, who famously instructed a graduate student in 1966 to "connect a camera to a computer and have it describe what it sees." Pity the kid: 50 years later, we're still working on it.

1.3.1.1 Vision

Reinventing the eye is the area where we've had the most success. Over the past few decades, we have created sensors and image processors that match and in some ways exceed the human eye's capabilities. With larger, more optically perfect lenses and semiconductor subpixels fabricated at nanometer scales, the precision and sensitivity of modern cameras is nothing short of incredible. Cameras can also record thousands of images per second and detect distances with great precision.

Yet despite the high fidelity of their outputs, these devices are in many ways no better than a pinhole camera from the 19th century: They merely record the distribution of photons coming in a given direction. The best camera sensor ever made couldn't recognize a ball — much less be able to catch it.

The hardware, in other words, is severely limited without the software — which, it turns out, is by far the greater problem to solve. But modern camera technology does provide a rich and flexible platform on which to work.

1.3.1.2 Recognition

This isn't the place for a complete course on visual neuroanatomy, but suffice it to say that our brains are built from the ground up with seeing in mind, so to speak. More of the brain is dedicated to vision than any other task, and that specialization goes all the way down to the cells themselves. Billions of them work together to extract patterns from the noisy, disorganized signal from the retina.

Sets of neurons excite one another if there's contrast along a line at a certain angle, say, or rapid motion in a certain direction. Higher-level networks aggregate these patterns into meta-patterns: a circle, moving upwards. Another network chimes in: the circle is white, with red lines. Another: it is growing in size. A picture begins to emerge from these crude but complementary descriptions.

Early research into computer vision, considering these networks as being unfathomably complex, took a different approach: "top-down" reasoning a book looks like this, so watch for this pattern, unless it's on its side, in which case it looks more like this. A car looks like this and moves like /this/.

We can barely come up with a working definition of how our minds work, much less how to simulate it.

For a few objects in controlled situations, this worked well, but imagine trying to describe every object around you, from every angle, with variations for lighting and motion and a hundred other things. It became clear that to achieve even toddler-like levels of recognition would require impractically large sets of data.

A "bottom-up" approach mimicking what is found in the brain is more promising. A computer can apply a series of transformations to an image and discover edges, the objects they imply, perspective and movement when presented with multiple pictures, and so on. The processes involve a great deal of math and statistics, but they amount to the computer trying to match the shapes it sees with shapes it has been trained to recognize trained on other images, the way our brains were.

What an image like this one above (from Purdue University's E-lab) is showing is the computer displaying that, by its calculations, the objects highlighted look and act like other examples of that object, to a certain level of statistical certainty.

Proponents of bottom-up architecture might have said “I told you so.” Except that until recent years, the creation and operation of artificial neural networks was impractical because of the immense amount of computation they require. Advances in parallel computing have eroded those barriers, and the last few years have seen an explosion of research into and using systems that imitate — still very approximately — the ones in our brain. The process of pattern recognition has been sped up by orders of magnitude, and we’re making more progress every day.

1.3.1.3 Understanding

Of course, you could build a system that recognizes every variety of apple, from every angle, in any situation, at rest or in motion, with bites taken out of it, anything — and it wouldn’t be able to recognize an orange. For that matter, it couldn’t even tell you what an apple is, whether it’s edible, how big it is or what they’re used for.

The problem is that even good hardware and software aren’t much use without an operating system.

For us, that’s the rest of our minds: short and long term memory, input from our other senses, attention and cognition, a billion lessons learned from a trillion interactions with the world, written with methods we barely understand to a network of interconnected neurons more complex than anything we’ve ever encountered.

The future of computer vision is in integrating the powerful but specific systems we’ve created with broader ones.

This is where the frontiers of computer science and more general artificial intelligence converge — and where we’re currently spinning our wheels. Between computer scientists, engineers, psychologists, neuroscientists and philosophers, we can barely come up with a working definition of how our minds work, much less how to simulate it.

That said, computer vision even in its nascent stage is still incredibly useful. It’s in our cameras, recognizing faces and smiles. It’s in self-driving cars, reading traffic signs and watching for pedestrians. There’s still a long way to go before they see like we do — if it’s even possible — but considering the scale of the task at hand, it’s amazing that they see at all.

1.3.1.4 Object Recognition

Object recognition is the area of artificial intelligence (AI) concerned with the abilities of robots and other AI implementations to recognize various things and entities.

Object recognition allows robots and AI programs to pick out and identify objects from inputs like video and still camera images. Methods used for object identification include 3D models, component identification, edge detection and analysis of appearances from different angles, Object recognition is at the convergence points of robotics, machine vision, neural networks and AI. Google and Microsoft are among the companies working in the area -- Google's driverless car and Microsoft's Kinect system both use object recognition.

Robots that understand their environments can perform more complex tasks better. Major advances of object recognition stand to revolutionize AI and robotics:

MIT has created neural networks, based on our understanding of how the brain works, that allow software to identify objects almost as quickly as primates do.

Gathered visual data from cloud robotics can allow multiple robots to learn tasks associated with object recognition faster. Robots can also reference massive databases of known objects and that knowledge can be shared among all connected robots.

Scientists at Brigham Young University have developed an object recognition algorithm that can learn to identify objects on its own. The Evolution-Constructed Features algorithm, as it's called, can make decisions about what characteristics of an object are relevant to its identification.

Concerns about the potential of object recognition include fears that advertisers and other interested entities will use the technology to mine the increasing number of images posted online and gather from them the personal information of individuals.

1.3.1.5 Computer Vision Application

i. Face detection

Popular applications include face detection and people counting. Have you ever noticed how Facebook detects your face when you upload a photo? This is a simple application of object detection that we see in our daily life.

ii. People Counting

Object detection can be also used for people counting, it is used for analyzing store performance or crowd statistics during festivals. These tend to be more difficult as people move out of the frame quickly (also because people are non-rigid objects).

iii. Vehicle detection

Similarly when the object is a vehicle such as a bicycle or car, object detection with tracking can prove effective in estimating the speed of the object. The type of ship entering a port can be determined by object detection (depending on shape, size etc.). This system for detecting ships are currently in development in some European countries.

iv. Manufacturing Industry

Object detection is also used in industrial processes to identify products. Say you want your machine to only detect circular objects. Hough circle detection transform can be used for detection.

v. Online images

Apart from these object detection can be used for classifying images found online. Obscene images are usually filtered out using object detection.

vi. Security

In the future we might be able to use object detection to identify anomalies in a scene such as bombs or explosives (by making use of a quadcopter).

1.4 Project Overview

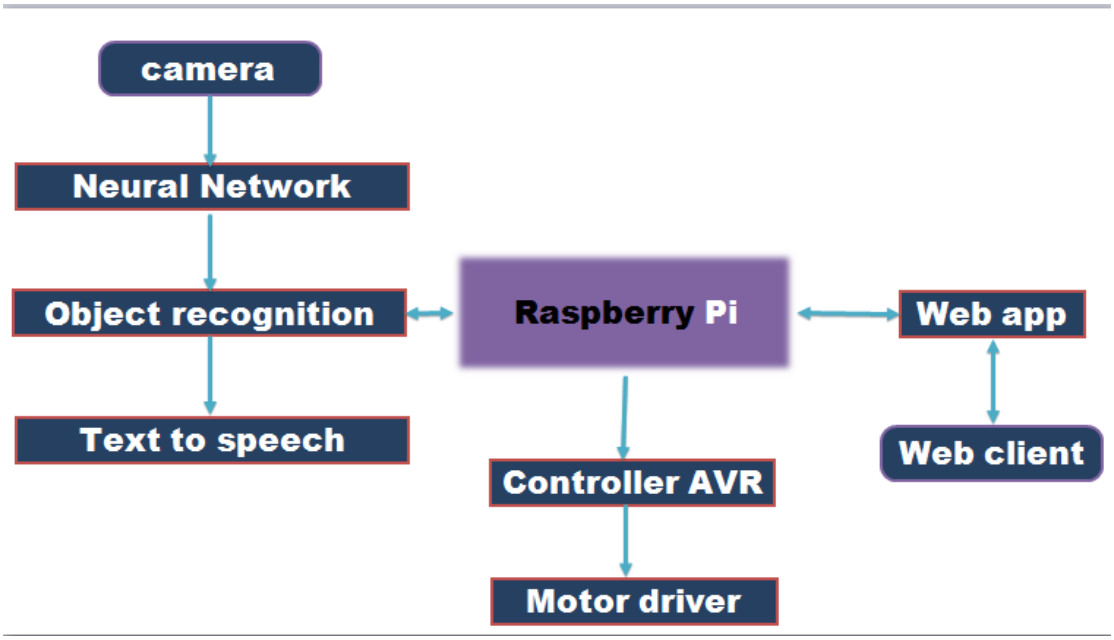


Figure 1: Project overview

Our project can be described as a mobile robot capable of objects detection and recognition, providing live stream through a web interface.

Robot characteristics:-

1. Multi MCU use with prober interfacing.
2. Object recognition using Inception model in google TensorFlow library.
3. Full remote control over the web granting great mobility.

CHAPTER 2:

Embedded System Control Unit

Broadly define an embedded system as a microcontroller-based, software-driven, reliable, real-time control system, designed to perform a specific task. It can be thought of as a computer hardware system having software embedded in it. An embedded system can be either an independent system or a part of a large system.

2.1 Embedded System

2.1.1 Definitions

2.1.1.1 System

A system is an arrangement in which all its unit assemble work together according to a set of rules. It can also be defined as a way of working, organizing or doing one or many tasks according to a fixed plan. For example, a watch is a time displaying system. Its components follow a set of rules to show time. If one of its parts fails, the watch will stop working. So we can say, in a system, all its subcomponents depend on each other.

2.1.1.2 Embedded System

As its name suggests, Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task. For example, a fire alarm is an embedded system; it will sense only smoke.

An embedded system has three components:
- It has hardware.
- It has application software.
- It has Real Time Operating system (RTOS) that supervises the application software and provide mechanism to let the processor run a process as per scheduling by following a plan to control the latencies.

RTOS defines the way the system works. It sets the rules during the execution of application program. A small scale embedded system may not have RTOS.

So we can define an embedded system as a Microcontroller based, software driven, and reliable, real-time control system.

2.1.2 Characteristics of an Embedded System

Single-functioned − An embedded system usually performs a specialized operation and does the same repeatedly. For example: A pager always functions as a pager.

Tightly constrained − All computing systems have constraints on design metrics, but those on an embedded system can be especially tight. Design metrics is a measure of an implementation's features such as its cost, size, power, and performance. It must be of a size to fit on a single chip, must perform fast enough to process data in real time and consume minimum power to extend battery life.

Reactive and Real time − Many embedded systems must continually react to changes in the system's environment and must compute certain results in real time without any delay. Consider an example of a car cruise controller; it continually monitors and reacts to speed and brake sensors. It must compute acceleration or de-accelerations repeatedly within a limited time; a delayed computation can result in failure to control of the car.

Microprocessors based − It must be microprocessor or microcontroller based.

Memory − It must have a memory, as its software usually embeds in ROM. It does not need any secondary memories in the computer.

Connected − It must have connected peripherals to connect input and output devices.

HW-SW systems − Software is used for more features and flexibility. Hardware is used for performance and security.

2.1.3 Basic Structure of an Embedded System

The following illustration shows the basic structure of an embedded system:

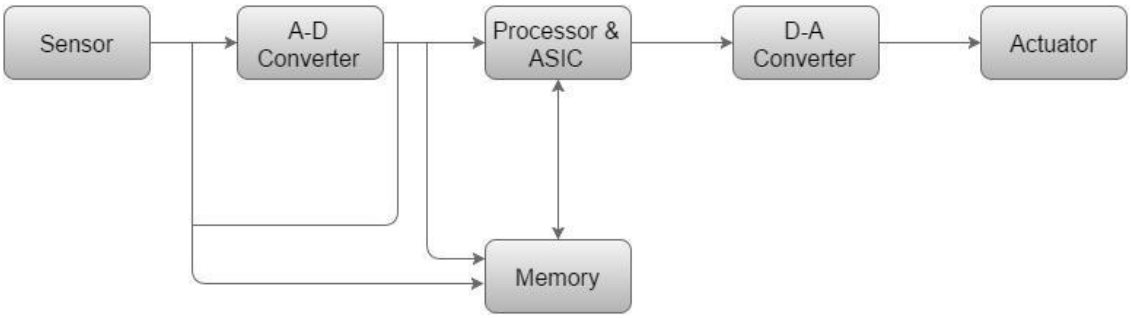


Figure 2: The basic structure of an embedded system

Sensor − It measures the physical quantity and converts it to an electrical signal which can be read by an observer or by any electronic instrument like an A2D converter. A sensor stores the measured quantity to the memory.

A-D Converter − An analog-to-digital converter converts the analog signal sent by the sensor into a digital signal.

Processor & ASICs − Processors process the data to measure the output and store it to the memory.

D-A Converter − A digital-to-analog converter converts the digital data fed by the processor to analog data

Actuator − An actuator compares the output given by the D-A Converter to the actual (expected) output stored in it and stores the approved output.

2.1.4 Processors

Processor is the heart of an embedded system. It is the basic unit that takes inputs and produces an output after processing the data. For an embedded system designer, it is necessary to have the knowledge of both microprocessors and microcontrollers.

A processor has two essential units:

- Program Flow Control Unit (CU)
- Execution Unit (EU)

The CU includes a fetch unit for fetching instructions from the memory. The EU has circuits that implement the instructions pertaining to data transfer operation and data conversion from one form to another.

The EU includes the Arithmetic and Logical Unit (ALU) and also the circuits that execute instructions for a program control task such as interrupt, or jump to another set of instructions.

A processor runs the cycles of fetch and executes the instructions in the same sequence as they are fetched from memory.

2.1.5 Microcontroller

A microcontroller is a single-chip VLSI unit (that is also called microcomputer), although it have limited computational capabilities, possesses enhanced input/output capability and a number of on-chip functional units.

Microcontrollers are particularly used in embedded systems for real-time control applications with on-chip program memory and devices.

2.1.5.1 How to choose microcontroller

Selecting the right microcontroller for a product can be a daunting task. Not only are there a number of technical features to consider, there are also business case issues such as cost and lead-times that can cripple a project. At the start of a project there is a great temptation to jump in and start selecting a microcontroller before the details of the system has been hashed out. This is of course a bad idea. Before any thought is given to the microcontroller, the hardware and software engineers should work out the high levels of the system, block diagram and flowchart them and only then is there enough information to start making a rational decision on microcontroller selection. When that point is reached, there are 10 easy steps that can be followed to ensure that the right choice is made.

- **Step 1: Make a list of required hardware interfaces**

Using the general hardware block diagram, make a list of all the external interfaces that the microcontroller will need to support. There are two general types of interfaces that need to be listed. The first are communication interfaces. These are peripherals such as USB, I2C, SPI, UART, and so on. Make a special note if the application requires USB or some form of Ethernet. These interfaces greatly affect how much program space the microcontroller will need to support. The second type of interface is digital inputs and outputs, analog to digital inputs, PWM's, etc. These

two interface types will dictate the number of pins that will be required by the microcontroller. Figure 3 shows a generic example of a block diagram with the i/o requirements listed.

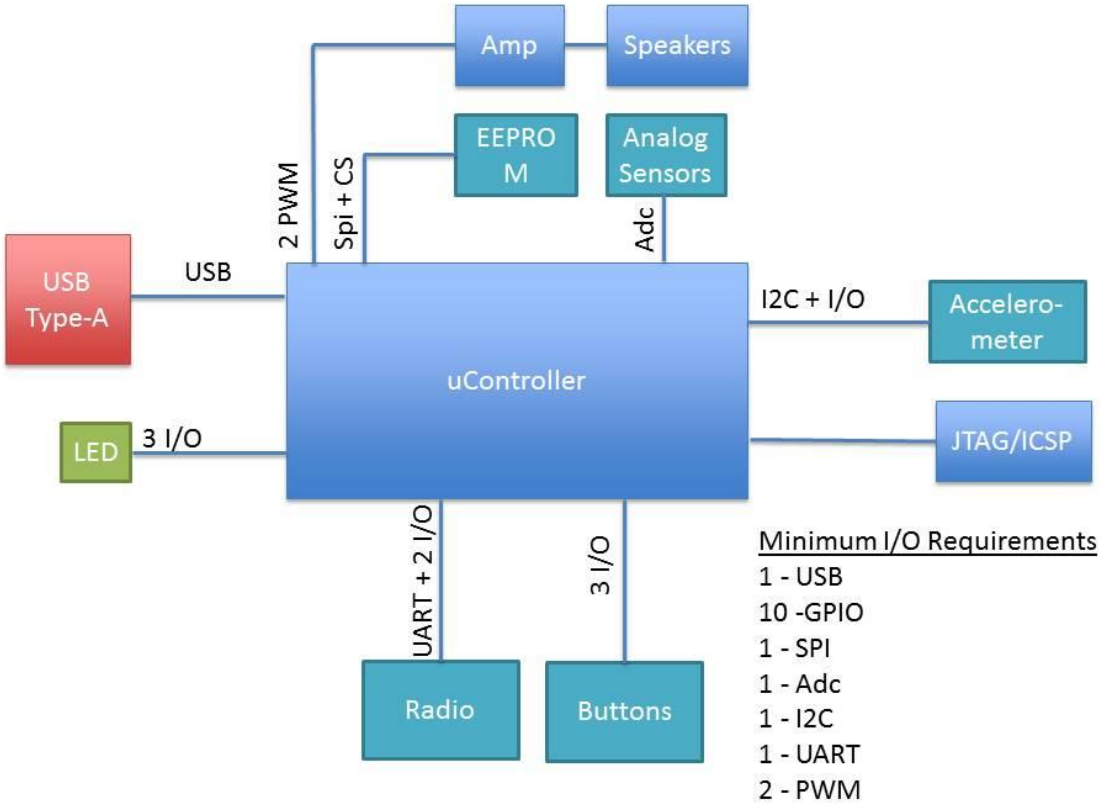


Figure 3: Generic example of a block diagram

- **Step 2: Examine the software architecture**

The software architecture and requirements can greatly affect the selection of a microcontroller. How heavy or how light the processing requirements will determine whether you go with an 80 MHz DSP or an 8 MHz 8051. Just like with the hardware, make notes of any requirements that will be important. For example, do any of the algorithms require floating point mathematics? Are there any high frequency control loops or sensors? Estimate how long and how often each task will need to run. Get an order of magnitude feel for how much processing power will be needed. The amount of computing power required will be one of the biggest requirements for the architecture and frequency of the microcontroller.

- **Step 3: Select the architecture**

Using the information from steps 1 and 2 an engineer should be able to start getting an idea of the architecture that will be needed. Can the application get by with eight bit architectures? How about 16 bits? Does it require a 32 bit ARM core? Between the application and the required software algorithms these questions will start to converge on a solution. Don't forget keep in mind possible future requirements and feature creep. Just because you could currently get by with an 8 bit microcontroller doesn't mean you shouldn't consider a 16 bit microcontroller for future features or even for ease of use. Don't forget that microcontroller selection can be an iterative process. You may select a 16-bit part in this step but then in a later step find that a 32 bit ARM part works better. This step is simply gets an engineer to look in the right direction.

- **Step 4: Identify Memory Needs**

Flash and RAM are two very critical components of any microcontrollers. Making sure that you don't run out of program space or variable space is undoubtedly of highest priority. It is far easier to select a part with too much of these features than not enough. Getting to the end of a design and discovering that you need 110% or that features need to be cut just isn't going to fly. After all, you can always start with more and then later move to a more constrained part within the same chip family. Using the software architecture and the communication peripherals included in the application, an engineer can estimate how much flash and RAM will be required for the application. Don't forget to leave room for feature creep and the next versions! It will save many headaches in the future.

- **Step 5: Start searching for microcontrollers**

Now that there is a better idea of what the required features of the microcontroller will be the search can begin! One place that can be a good place to start is with a microcontroller supplier such as Arrow, Avnet, Future Electronics or similar. Talk with an FAE about your application and requirements and often times they can direct you to a new part that is cutting edge and meets the requirements. Just keep in mind that they might have pressure on them at that time to push a certain family of microcontrollers!

The next best place to start is with a silicon provider that you are already familiar with. For example, if you have used Microchip parts in the past and had a good experience with them, then start at their website. Most silicon providers have a search engine that allows you to enter your peripheral sets, I/O and power requirements and it will narrow down the list of parts that match the criteria. From that list the engineer can then move forward towards selecting a microcontroller.

- **Step 6: Examine Costs and Power Constraints**

At this point the selection process has revealed a number of potential candidates. This is a great time to examine the power requirements and cost of the part. If the device will be powered from a battery and mobile, then making sure the parts are low-power is absolutely precarious. If it doesn't meet power requirements then keep weeding the list down until you have a select few. Don't forget to examine the piece price of the processor either. While prices have steadily been approaching $1 in volume for many parts, if it is highly specialized or a high-end processing machine then price might be critical. Don't forget about this key element.

- **Step 7: Check part availability**

With the list of potential parts in hand, now is a good time to start checking on how available the part is. Some of the things to keep in mind are what the lead times for the part? Are they kept in stock at multiple distributors or is there 6 – 12 week lead time? What are your requirements for availability? You don't want to get stuck with a large order and have to wait three months to be able to fill it. Then there is a question of how new the part is and whether it will be around for the duration of your product life cycle. If your product will be around for 10 years then you need to find a part that the manufacturer guarantees will still be built in 10 years.

- **Step 8: Select a development kit**

One of the best parts of selecting a new microcontroller is finding a development kit to play with and learn the inner working of the controller. Once an engineer has settled their heart on the part they want to use they should research what development kits are available. If a development kit isn't available then the selected part is most likely not a good choice and they should go back a few steps and find a better part. Most development kits today cost under $100. Paying any more than that (unless it is designed to work with multiple processor modules) is just too much. Another part may be a better choice.

- **Step 9: Investigate compilers and tools**

The selection of the development kit nearly solidifies the choice of microcontroller. The last consideration is to examine the compiler and tools that are available. Most microcontrollers have a number of choices for compilers, example code and debugging tools. It is important to make sure that all the necessary tools are available for the part.

- **Step 10: Start Experimenting**

Even with the selection a microcontroller nothing is set in stone. Usually the development kit arrives long before the first prototyped hardware. Take advantage by building up test circuits and interfacing them to the microcontroller. Choose high risk parts and get them working on the development kit. It may be that you discover the part you thought would work great has some unforeseen issue that would force a different microcontroller to be selected.

2.1.5.2 Comparison between different microcontrollers

Here are the main difference between AVR, ARM, 8051 and PIC which are the most well-known microcontrollers:

	8051	**PIC**	**AVR**	**ARM**
Bus width	8-bit for standard core	8/16/32-bit	8/32-bit	32-bit mostly also available in 64-bit
Communic-ation Protocols	UART, USART,SPI,I2C	PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S	UART, USART, SPI, I2C, (special purpose AVR support CAN, USB, Ethernet)	UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI (serial audio interface), IrDA

Speed	12 Clock/instruction cycle	4 Clock/instruction cycle	1 clock/ instruction cycle	1 clock/ instruction cycle
Memory	ROM, SRAM, FLASH	SRAM, FLASH	Flash, SRAM, EEPROM	Flash, SDRAM, EEPROM
ISA	CISC	Some feature of RISC	RISC	RISC
Memory Architecture	Von Neumann architecture	Harvard architecture	Modified	Modified Harvard architecture
Power Consumption	Average	Low	Low	Low
Families	8051 variants	PIC16,PIC17, PIC18, PIC24, PIC32	Tiny, Atmega, Xmega, special purpose AVR	ARMv4,5,6,7 and series
Community	Vast	Very Good	Very Good	Vast
Manufacturer	NXP, Atmel, Silicon Labs, Dallas, Cyprus, Infineon, etc.	Microchip Average	Atmel	Apple, Nvidia, Qualcomm, Samsung Electronics, and TI etc.
Cost (compared to features provide)	Very Low	Average	Average	Low
Other Feature	Known for its Standard	Cheap	Cheap, effective	High speed operation Vast

Popular Microcontrollers	AT89C51, P89v51, etc.	PIC18fXX8, PIC16f88X, PIC32MXX	Atmega8, 16, 32, Arduino Community	LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc.

Table 1 : Comparison between u-controllers

2.2 Raspberry Pi

As significant advances in computing go, the Raspberry Pi's primary innovation was the lowering of the entry barrier into the world of embedded Linux. The barrier was twofold price and complexity. The Raspberry Pi's low price solved the price problem (cheap is good!) and the SoC reduced circuit complexity rather dramatically, making a much smaller package possible. The road to the development of the Raspberry Pi originated at a surprising point through a registered charity in the UK, which continues to operate today. The Raspberry Pi Foundation, registered with the Charity Commission for England and Wales, first opened its doors in 2009 in Caldecott, Cambridge shire. It was founded for the express purpose of promoting the study of computer science in schools. A major impetus for its creation came from a team consisting of Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft. At the University of Cambridge's Computer Laboratory, they had noted the declining numbers and low-level skills of student applicants. They came to the conclusion that a small, affordable computer was needed to teach basic skills in schools and to instill enthusiasm for computing and programming. Major support for the Foundation's goals came from the University of Cambridge Computer Laboratory and Broadcom, which is the company that manufactures the SoC the Broadcom 2835 or 2836, depending on the model that enables the Raspberry Pi's power and success. Later in this chapter you will read more on that component, which is the heart and soul of the Raspberry Pi. The founders of the Raspberry Pi had identified and acted on the perceived need for a tiny, affordable computer. By 2012, the Model B had been released at a price of about £25. The fact that this represented great value for money was recognized immediately, and first-day sales blasted over 100,000 units. In less than two years of production, more than two million boards were sold. The Raspberry Pi continued to enjoy good sales and wide acceptance following the highly successful release of the Model B+ (in late 2014). And in 2015, the fast, data-crunching Raspberry Pi 2 Model B with its four-core ARM processor and additional onboard memory sold more than 500,000 units in its first two weeks of release. Most recently, the Raspberry Pi Zero, a complete computer system on a board for £4 yes, £4 was released. It's an awesome deal if you can get one—the first batch sold out almost immediately. In 2016, the Raspberry Pi Model 3 Model B arrived. It supports a 1.2GHz 64-bit quad-core ARMv8 CPU, 1 GB RAM, and built-in wireless and

Bluetooth! All for the same low price. The original founders of the Raspberry Pi Foundation included:

- Eben Upton
- Rob Mullins
- Jack Lang
- Alan Mycroft
- Pete Lomas
- David Braben

The organization now consists of two parts:

- Raspberry Pi (Trading) Ltd. performs engineering and sales, with Eben Upton as CEO.
- The Raspberry Pi Foundation is the charitable and educational part.

The idea behind a tiny and affordable computer for kids came in 2006, when Eben Upton, Rob Mullins, Jack Lang and Alan Mycroft, based at the University of Cambridge's Computer Laboratory, became concerned about the year-on-year decline in the numbers and skills levels of the A Level students applying to read Computer Science. From a situation in the 1990s.

Where most of the kids applying were coming to interview as experienced hobbyist programmers, the landscape in the 2000s was very different; a typical applicant might only have done a little web design.

As a result, the founders' stated goal was "to advance the education of adults and children, particularly in the field of computers, computer science and related subjects". Their answer to the problem, of course, was the Raspberry Pi, which was designed to emulate in concept the hands-on appeal of computers from the previous decade (the 1990s). The intention behind the Raspberry Pi was to be a "catalyst" to inspire students by providing affordable, programmable computers everywhere. The Raspberry Pi is well on its way to achieving the Foundation's goal in bettering computer education for students. However, another significant thing has happened; a lot of us older people have found the Raspberry Pi exciting. It's been adopted by generations of hobbyists, experimenters and many others, which has driven sales into new millions of units. While the sheer compactness of the Raspberry Pi excites, resonates and inspires adults as well as youngsters, what truly prompted its success was its low price and scope of development. Embedded Linux has always been a painful subject to learn, but the Pi makes it simple and inexpensive.

2.2.1 Raspberry Pi Versions

Figure 2-1 provides a summary feature comparison of the different RPi models that are presently available. Here is a quick summary of this table:

- If you need an RPi for general-purpose computing, consider the RPi 3. The 1 GB of memory and 1.2 GHz quad-core processor provide the best performance out of all the boards.
- For applications that interface electronics circuits to the Internet on a wired network, consider the RPi 3, RPi 2, or RPi B+, with cost being the deciding factor.
- If you need a small-footprint device with wireless connectivity, consider the RPi Zero. The RPi A+ could be used to develop the initial prototype.
- If you want to design your own PCB that uses the RPi (or multiple RPi boards), investigate the Compute module.

	Raspberry Pi 3 Model B	Raspberry Pi Zero	Raspberry Pi 2 Model B	Raspberry Pi Model B+
Introduction Date	2/29/2016	11/25/2015	2/2/2015	7/14/2014
SoC	BCM2837	BCM2835	BCM2836	BCM2835
CPU	Quad Cortex A53 @ 1.2GHz	ARM11 @ 1GHz	Quad Cortex A7 @ 900MHz	ARM11 @ 700MHz
Instruction set	ARMv8-A	ARMv6	ARMv7-A	ARMv6
GPU	400MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV	250MHz VideoCore IV
RAM	1GB SDRAM	512 MB SDRAM	1GB SDRAM	512MB SDRAM
Storage	micro-SD	micro-SD	micro-SD	micro-SD
Ethernet	10/100	none	10/100	10/100
Wireless	802.11n / Bluetooth 4.0	none	none	none
Video Output	HDMI / Composite	HDMI / Composite	HDMI / Composite	HDMI / Composite
Audio Output	HDMI / Headphone	HDMI	HDMI / Headphone	HDMI / Headphone
GPIO	40	40	40	40
Price	$35	$5	$35	$35

Table 2: Comparison between Pi models

2.2.2 The Raspberry Pi Hardware

Figure 5: Raspberry Pi 3 kit

2.2.2.1 Dimensions

The Raspberry Pi 3 is a small machine measuring only 85.60 mm x 56 mm x 21 mm and weighing approximately 45g. This small size makes it suitable for embedded projects, home automation devices, arcade machines, or building small multi-device clusters.

2.2.2.2 System on Chip

An SoC or system-on-a-chip is an integrated circuit (IC) that has the major components of a computer or any other electronic system on a single chip. The components include a central processing unit (CPU), a graphics processing unit (GPU) and various digital, analogue and mixed signal circuits on just one chip.

This SoC component makes highly dense computing possible, such as all the power that is shoehorned into the Raspberry Pi. The Raspberry Pi features chips that are developed and manufactured by Broadcom Limited. Specifically, the older models as well as the latest (the £4 Raspberry Pi Zero) come with the Broadcom BCN2835 and the Raspberry Pi 2 has the Broadcom BCM2836, and the new Model 3 uses the Broadcom BCM2837. The biggest difference between these two SoC ICs is the replacement of the single-core CPU in the BCM2835 with a four-core processor in the BCM2836. Otherwise, they have essentially the same architecture.

Here's a taste of the low-level components, peripherals and protocols provided by the Broadcom SoC:

- **CPU:** Performs data processing under control of the operating system (a CPU with a single core on most of the Raspberry Pi models and a CPU with four cores on the Raspberry Pi 2 and Raspberry Pi 3).
- **GPU:** Provides the operating system desktop.
- **Memory:** Permanent memory used as registers for CPU and GPU operation, storage for bootstrap software, the small program which starts the process of loading the operating system and activating it.
- **Timers:** Allow software to be time-dependent for scheduling, synchronizing and so on.
- **Interrupt controller:** Interrupts allow the operating system to control all the computer resources, know when the CPU is ready for new instructions and much more.
- **General purpose input output (GPIO):** Provides layout and enables control of connections, input, output and alternative modes for the GPIO pins that enable the Raspberry Pi to manage circuits, devices, machines and so on. In short, it turns the Raspberry Pi into an embeddable control system.
- **USB:** Controls the USB services and provides the Universal Serial Bus protocols for input and output, thus allowing peripherals of all types to connect to the Raspberry Pi's USB receptacles.
- **PCM/I2S:** Provides pulse code modulation (PCM, which converts digital sound to analogue sound such as speakers and headphones require) and known as Inter-IC Sound, Integrated Interchip Sound, or IIS, a high-level standard for connecting audio devices).
- **Direct memory access (DMA) controller:** Direct memory access control that allows an input/output device to bypass the CPU and send or receive data directory to the main memory for purposes of speed and efficiency.
- **I2C/SPI (Serial Peripheral Interface) slave:** The reverse of the preceding bullet point. Allows outside chips and sensors to control or cause the Raspberry Pi to respond in certain ways; for example, a sensor in a motor detects it's running hot and the controller chip causes the Raspberry Pi to make a decision on whether to reduce the motor's speed or stop it.
- **SPI Interface:** Serial interfaces, accessed via the GPIO pins and allowing the daisy chaining of several compatible devices by the use of different chip-select pins.

- **Pulse width modulation (PWM):** A method of generating an analogue waveform from a digital signal.
- **Universal asynchronous receiver/transmitter (UART0, UART1):** Used for serial communication between different devices.

2.2.2.3 MicroSD card port

The microSD card is the main boot and storage mechanism of the Raspberry Pi. It is upon the microSD card that you will load your operating system and store data. Later we will look at using the microSD purely for booting the Raspberry Pi, and then using a USB hard drive as a storage mechanism. In this chapter, we will dive into how we can setup the SD card with the Raspbian operating system.

2.2.2.4 Ethernet port

The Raspberry Pi 3 supports 10/100 Mbps Ethernet, and the USB adapter in the third/fourth port of USB hub can also be used for Ethernet via a USB to Ethernet adapter.

2.2.2.5 Audio Out

On the bottom of the board is the 3.5 millimeter (mm) audio input/output jack. Here you can plug in headphones, a computer sound card, speakers or anything thing else that takes and plays audio input.

2.2.2.6 Composite Video

Using the same 3.5mm socket described in the last section, old-style composite video is also available. When it boots up and finds a composite video device attached, the Raspberry Pi attempts to select the right resolution. Mostly it gives a usable display but sometimes it gets things wrong. Having video composite output may seem old school in light of the modern era's profusion of HDMI devices hanging off every wall, but it fits in with the design philosophy Raspberry Pi Foundation co-founder Eben Upton recently described. He said, "It's a very cheap Linux PC device in the spirit of the 1980s, a device which turns your TV into a computer; plug in to TV, plug a mouse and a keyboard in, give it some power and some kind of storage, an operating system and you've got a PC".

2.2.2.7 HDMI

There's nothing as fine as a nice big display showing the colorful graphical user interface (GUI) of the Raspberry Pi. A display enables you to surf the web, watch videos, and play games all the stuff you expect a computer to do. The best solution for that involves HDMI. High-Definition Multimedia Interface (HDMI) allows the transfer of video and audio from an HDMI-compliant display controller (in our case, the Raspberry Pi) to compatible computer monitors, projectors, digital TVs or digital audio devices. HDMI's higher quality provides a marked advantage over composite video (such as what comes out of the audio socket on the Raspberry Pi board). It's much easier on the eyes and provides higher resolution instead of composite video's noisy and sometimes distorted video. The HDMI connector on the Raspberry Pi Model B is approximately centered on the lower edge of the Raspberry Pi board.

2.2.2.8 CSI Camera Module Connector

Camera modules for the Raspberry Pi give you 5-megapixel stills and 1080 high-definition video for about £16. The Camera Serial Interface (CSI) connector located between the HDMI socket and the 3.5mm audio socket provides a place to plug the camera module into the Pi.

2.2.2.9 GPIO pins

The main method for interacting with electronic components and expansion boards is through the general purpose input/output (GPIO) pins on the Raspberry Pi. The Raspberry Pi 3 Model B contains 40 pins in total.

2.2.2.10 Micro USB Power

The micro USB cable supplies 5VDC to the Raspberry Pi at about 1 ampere (1A) of current for the model B. Some recommendations for the B+ mention 1.5A, but if you're pushing heavy current through the USB ports (remember, four instead of two on the B+ and later), a 2A supply is smarter. For the Raspberry Pi 2, get at least a 2.4A supply. Remember, there's no switch for turning the Raspberry Pi on and off (another saving to keep the price down). You just plug and unplug the micro USB connector. Of course, with a bit of tinkering and soldering, you could add a switch to the power cable easily enough.

2.2.2.11 DSI Display Connection

Just right of the SD card slot but on top of the board is the Display Serial Interface (DSI) display connector. The DSI connector's design accommodates a flat 15-conductor cable that drives liquid crystal display (LCD) screens.

2.2.3 Who Should Use the RPi

Anybody who wants to transform an engineering concept into a real interactive electronics project, prototype, or work of art should consider using the RPi. That said, integrating high-level software and low-level electronics is not an easy task. However, the difficulty involved in an implementation depends on the level of sophistication that the project demands. The RPi community is working hard to ensure that the platform is accessible by everyone who is interested in integrating it into their projects, whether they are students, makers, artists, or hobbyists. For example, the availability of the Scratch visual programming tool on the RPi (tiny.cc/erpi101) is an excellent way to engage children with both computer programming and the RPi. For more advanced users with electronics or computing knowledge, the RPi platform enables additional development and customization to meet specific project needs. Again, such customization is not trivial: You may be an electronics expert, but high-level software programming and/or the Linux operating system might cause you difficulty. Or you may be a programming guru but you have never wired an LED!.

2.2.4 What does the Raspberry Pi do

The Raspberry Pi excels as the brains for all sorts of projects. Here are some examples randomly picked from the many thousands of documented projects on the Internet. This list may inspire you in choosing some projects of your own:

- Home automation
- Home security
- Media centre
- Weather station
- Wearable computer
- Robot controller
- Web camera controller
- Coffee maker
- Ham radio EchoLink server
- Electric motor controller
- Time-lapse photography manager
- Game controller

- Quadcopter (drone) controller
- Bitcoin mining
- Web server
- Automotive onboard computer
- Email server
- GPS tracker

This list just scratches the surface of possible uses for the Raspberry Pi. There's not enough room to list everything you could do.

2.2.5 RPi Documentation

The Raspberry Pi Foundation website: This provides the main support for the RPi platform, with blogs, software guides, community links, and downloads to support your development. See www.raspberrypi.org. A huge amount of documentation is available on the RPi platform, but the most important documents for this book are as follows:

The Raspberry Pi Documentation: This is the official documentation for the RPi that is written by the Raspberry Pi Foundation. It includes guides on getting started, configuration, guides to Linux distributions, and more. See www.raspberrypi.org/documentation/.

Broadcom BCM2835 ARM Peripherals Datasheet: This is the core document that describes the processor on most RPi models. It is 200 pages long and provides a technical description of the functionality and capabilities of the processor on the RPi. See tiny.cc/erpi103. There is also an important errata document at tiny.cc/erpi104.

The BCM2836 Document: This document describes features of the processor on the RPi 2, and related features on the RPi 3. It should be read in association with the previous Broadcom document for the BCM2835. See tiny.cc/erpi105.

2.2.6 How to Destroy Your RPi

RPi boards are complex and delicate devices that are very easily damaged if you do not show due care. If you are moving up from boards like the Arduino to the RPi platform, you have to be especially careful when connecting circuits that you built for that platform to the RPi. Unlike the Arduino Uno, the microprocessor on the RPi

cannot be replaced. If you damage the microprocessor SoC, you will have to buy a new board!

- Do not shut the RPi down by pulling out the USB power supply. You should shut down the board correctly using a software shutdown procedure.
- Do not place a powered RPi on metal surfaces (e.g., aluminum-finish computers) or on worktops with stray/cut-off wire segments, resistors, etc. If you short the pins underneath the GPIO header, you can easily destroy your board. Alternatively, you can attach small self-adhesive rubber feet to the bottom of the RPi.
- Do not connect circuits that source/sink other than very low currents from/to the GPIO header. The maximum current that you can source or sink from many of these header pins is approximately 2 mA to 3 mA. The power rail and ground pins can source and sink larger currents. For comparison, some Arduino models allow currents of 40 mA on each input/output.
- The GPIO pins are 3.3 V tolerant. Do not connect a circuit that is powered at 5 V; otherwise, you will destroy the board.
- Do not connect circuits that apply power to the GPIO header while the RPi is not powered on. Make sure that all self-powered interfacing circuits are gated by the 3.3 V supply line or using opto-couplers.
- Carefully check the pin numbers that you are using. There are 40 pins on the GPIO header, and it is very easy to plug into header connector 21 instead of 19. The T-Cobbler board is very useful for interconnecting the RPi to a breadboard, and it is highly recommended for prototyping work.

2.2.7 Getting Started With Raspberry Pi

In order to get up and running with your Raspberry Pi 3 you will need the following additional hardware components:

• MicroSD card

• Micro USB power cable

• Monitor preferably HDMI

• HDMI cable or 3.5mm to RCA AV cable

• USB keyboard

• USB mouse

• Protective case optional

• Wi-Fi dongle or Ethernet cable

2.2.7.1 Downloading the latest version of Raspbian

Your first task will be to download the Raspbian operating system from the official Raspberry Pi website at https://www.raspberrypi.org/downloads/raspbian/.

2.2.7.2 Setting up your microSD card and installing

The Raspbian installation process involves two steps:

• Formatting the microSD card to the FAT file system

• Copying the Raspbian image to the card

It is important that we quickly look at what File Allocation Table (FAT) is and why we need it. FAT is a method for defining which sectors of a disk or microSD card files are stored in and which sectors on the disk are free to have new data written to them. The standard has its origins in the 1970s for use on floppy disks and was developed by Bill Gates and Marc McDonald.

Due to its simplicity of implementation and robustness, this standard is still used on SD and microSD cards today. Therefore, it is the format you will need in order to install the Raspberry Pi's operating system onto your microSD card. Due to its widespread adoption you may find and microSD card you purchase is already formatted to FAT. We recommend, however, formatting any new cards you purchase to ensure you do not encounter any problems. The official Raspberry Pi website provides handy how-to guides for the three major operating systems on how to format and install the Raspbian image. You can read an up-to-date overview of the installation procedure at https://www.raspberrypi.org/documentation/installation/installing-images/README.md

Having completed installing the operating system we can now look at some final configuration before exploring some interesting features of Raspbian

2.2.7.3 Raspbian installation wrap-up

The following section assumes you have your Raspberry Pi connected to a monitor and with a keyboard and mouse available. It also assumes you have your configuration set to boot to desktop and have powered up and logged into your device. You should at this point connect your device to your home router. If you are planning on using Wi-Fi, read on. Now that you have successfully installed Raspbian you should see the Linux desktop.

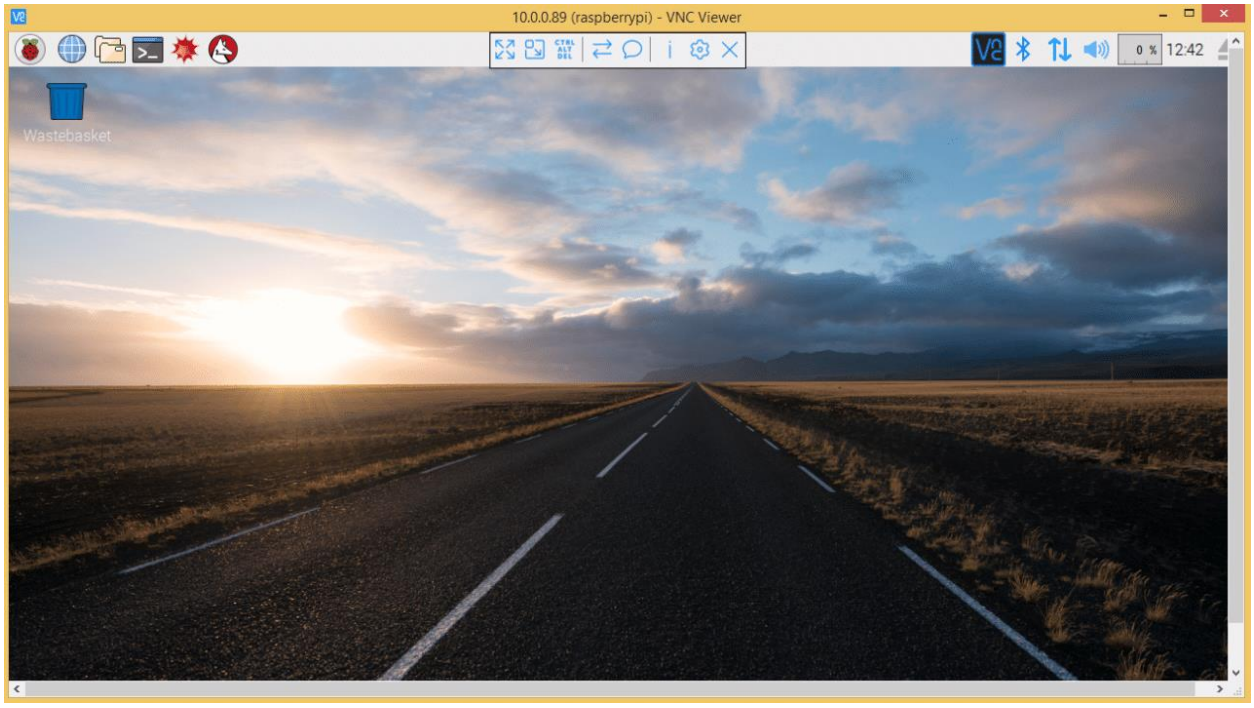


Figure 6: Raspbian OS GUI

This desktop contains icons in the top menu linking to a number of programs installed by default with the operating system. One important icon is the link to LXTerminal. This icon launches the Linux terminal window. Click on this icon and you should see the command line load. The following tasks in this section can all be performed in this window.

As a handy shortcut you can also load the raspi-config application at any time by typing the following command:

sudo raspi-config

If you update settings in this manner you may need to reboot the Raspberry Pi for them to take affect.

2.2.7.4 Check SSH is running

In order to connect to our Raspberry Pi 3 from another device via a terminal window we need to ensure that the Secure Shell(SSH) server is up and running. SSH is the default mechanism for secure communication between our Linux machines. If you used NOOBS to install the OS you may have configured the SSH server at this point via the advanced options. We can check that the SSH service is running successfully as follows.

Open up a terminal window from the Raspbian desktop and type the following command:

ps aux | grep sshd

The following sshd process should be displayed. This tells us the services are up and running:

root 2017 0.0 0.3 6228 2892 ? Ss 15:13 0:00 /usr/sbin/ sshd

If the SSH process does not appear, it is simple to start it. Enter the following command into the terminal:

sudo /etc/init.d/ssh start

After you have executed this command try running the following again and check that the sshd process is now running:

ps aux | grep sshd

By default, to login to the Raspberry Pi 2 over SSH you will be prompted for a username and password. If you have not changed this the username is pi and the password is raspberry.

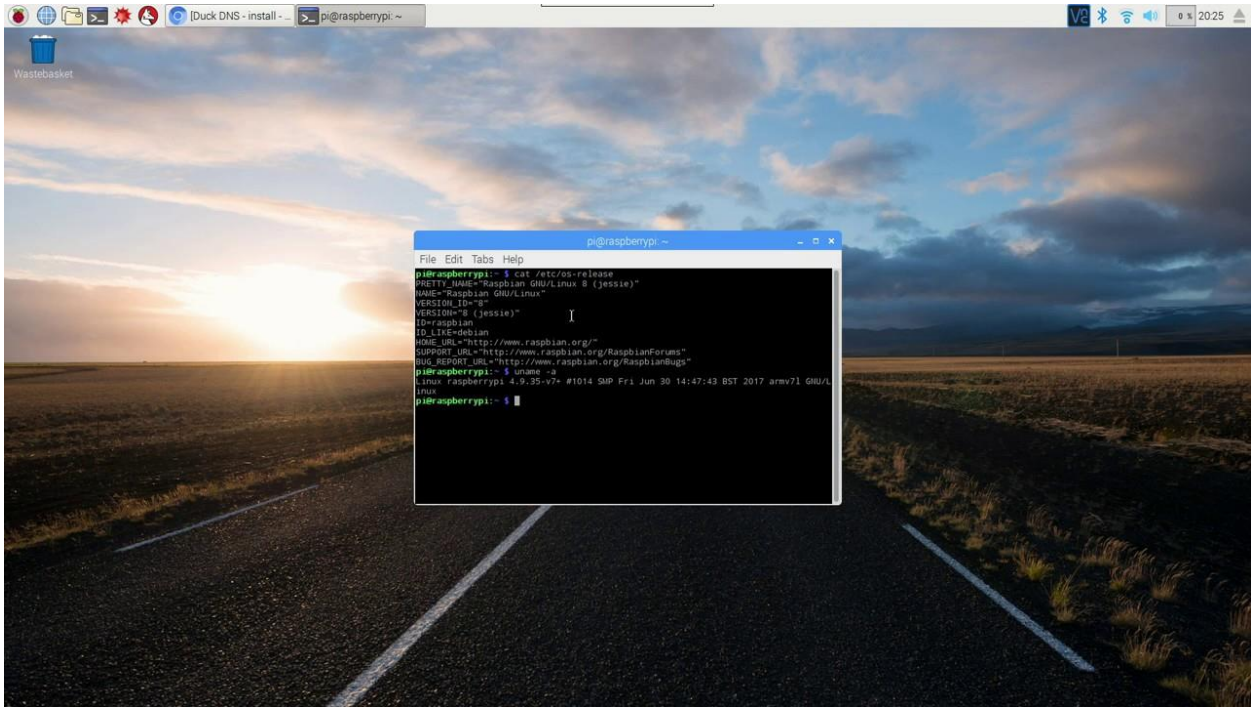


Figure 7: Bash Terminal

2.2.8 Programming on pi

There is a plethora of programming languages available on the Raspberry Pi, so knowing where to start can be hard. Many languages are useful for a variety of different project types, including building websites, programming hardware, and writing desktop applications.

2.2.8.1 Python

Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

I - Starting Python

The Raspberry Pi has two versions of Python installed on it: Python 2.7 and Python 3. Usually when software or programming languages are updated, the new version is compatible with the old version. Python 3 was intentionally designed not to be compatible, however, so programs written for Python 2.7 might not work with Python 3, and vice versa.

II - Entering Your First Python Commands

When you start IDLE, a window opens, this is the Python shell, and the three arrows are your prompt, which means Python is ready for you to enter a command. You can test this by entering the license() command, which shows you a history of Python before displaying the terms and conditions of using it. If you don't want to get bogged down in legalese, abort by pressing q and then pressing Enter when prompted. One of the most basic commands in any programming language is the one that tells the computer to put some text on the screen. In Python (and some other languages too), this command is print, and you use it like this:

>>> print "Hello, world"

Hello, world

>>>

Whatever you type in the quotes after the print command is "printed" on the screen, and Python then returns you to the prompt so you can enter another command.

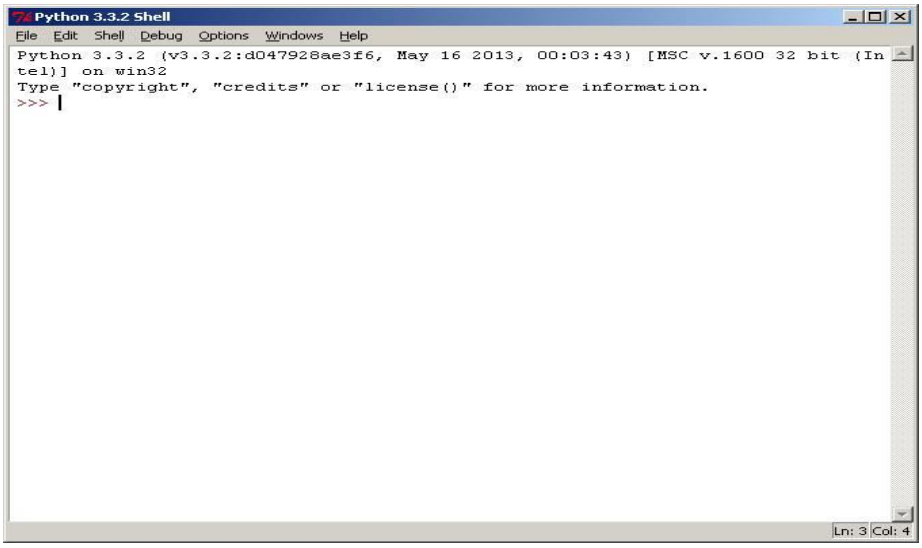


Figure 8: Python Shell

2.3 Interfacing between controllers

The used microcontrollers are supposed to interact between each other and this can be achieved by several ways, You may even do that by direct hard wired connection between them, Yet this wouldn't be the most efficient way at the point comes the communication protocols.

2.3.1 Why Use I2C?

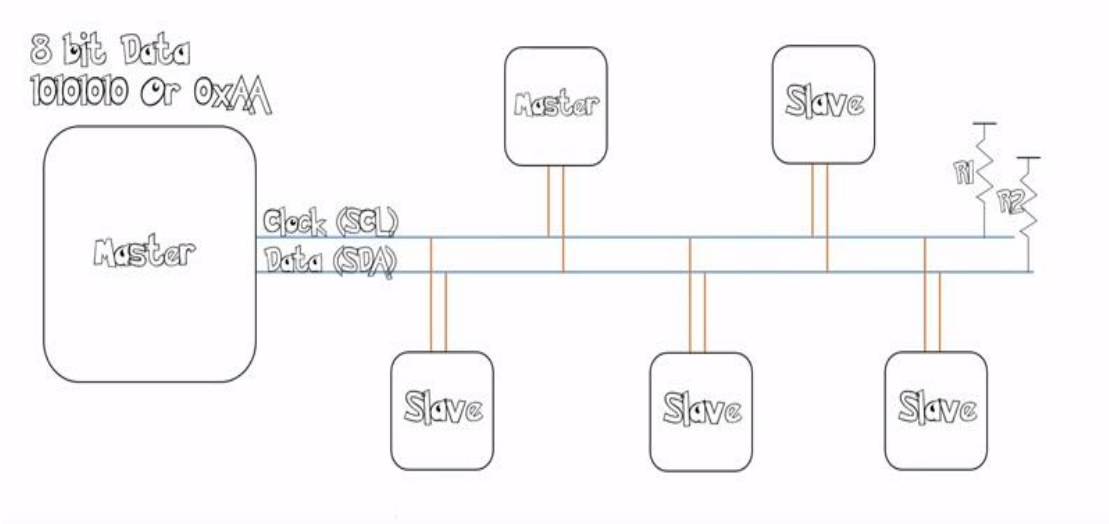


Figure 9: TWI Connections

To figure out why one might want to communicate over I2C, you must first compare it to the other available options to see how it differs.

2.3.1.1 What's wrong with Serial Ports?

Because serial ports are **asynchronous** (no clock data is transmitted), devices using them must agree ahead of time on a data rate. The two devices must also have clocks that are close to the same rate, and will remain so–excessive differences between clock rates on either end will cause garbled data.
Asynchronous serial ports require hardware overhead–the UART at either end is relatively complex and difficult to accurately implement in software if necessary. At least one start and stop bit is a part of each frame of data, meaning that 10 bits of transmission time are required for each 8 bits of data sent, which eats into the data rate.
Another core fault in asynchronous serial ports is that they are inherently suited to communications between two, and only two, devices. While it is possible to connect multiple devices to a single serial port, **bus contention** (where two devices attempt to

drive the same line at the same time) is always an issue and must be dealt with carefully to prevent damage to the devices in question, usually through external hardware.

2.3.1.2 What's wrong with SPI?

The most obvious drawback of SPI is the number of pins required. Connecting a single master to a single slave with an SPI bus requires four lines; each additional slave requires one additional chip select I/O pin on the master. The rapid proliferation of pin connections makes it undesirable in situations where lots of devices must be slaved to one master. Also, the large number of connections for each device can make routing signals more difficult in tight PCB layout situations.
SPI only allows one master on the bus, but it does support an arbitrary number of slaves (subject only to the drive capability of the devices connected to the bus and the number of chip select pins available).
SPI is good for high data rate **full-duplex** (simultaneous sending and receiving of data) connections, supporting clock rates upwards of 10MHz (and thus, 10 million bits per second) for some devices, and the speed scales nicely. The hardware at either end is usually a very simple shift register, allowing easy implementation in software.

2.3.1.3 Enter I2C - The Best of Both Worlds!

I2C requires a mere two wires, like asynchronous serial, but those two wires can support up to 1008 slave devices. Also, unlike SPI, I2C can support a multi-master system, allowing more than one master to communicate with all devices on the bus (although the master devices can't talk to each other over the bus and must take turns using the bus lines).
Data rates fall between asynchronous serial and SPI; most I 2C devices can communicate at 100kHz or 400kHz. There is some overhead with I2C; for every 8 bits of data to be sent, one extra bit of meta data (the "ACK/NACK" bit, which we'll discuss later) must be transmitted.
The hardware required to implement I2C is more complex than SPI, but less than asynchronous serial. It can be fairly trivially implemented in software.

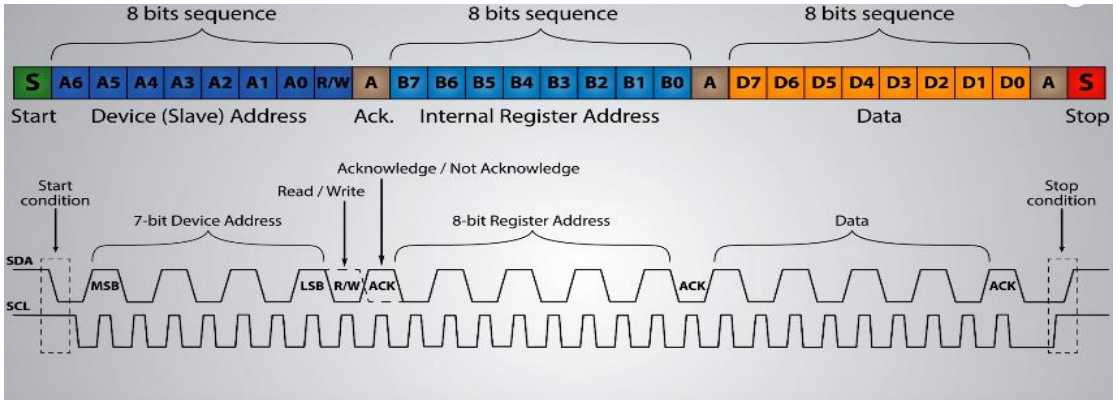


Figure 10: I2C Data Frame

2.3.2 I2C at the Hardware Level

Signals

Each I2C bus consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master; some slave devices may force the clock low at times to delay the master sending more data (or to require more time to prepare data before the master attempts to clock it out). This is called “clock stretching”, the I2C bus drivers are “open drain”, meaning that they can pull the corresponding signal line low, but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system. Each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low.
Resistor selection varies with devices on the bus, but a good rule of thumb is to start with 4.7k and adjust down if necessary. I2C is a fairly robust protocol, and can be used with short runs of wire (2-3m).
For long runs, or systems with lots of devices, smaller resistors are better.

Signal Levels

Since the devices on the bus don’t actually drive the signals high, 2 IC allows for some flexibility in connecting devices with different I/O voltages. In general, in a system where one device is at a higher voltage than another, it may be possible to connect the two devices via I2C without any level shifting circuitry in between them. The trick is to connect the pull-up resistors to the lower of the two voltages. This only works in some cases, where the lower of the two system voltages exceeds the high-level input voltage of the higher voltage system–for example, a 5V Arduino and a 3.3V accelerometer.

Protocol

Communication via I2C is more complex than with a UART or SPI solution. The signaling must adhere to a certain protocol for the devices on the bus to recognize it as valid I2C communications.
Fortunately, most devices take care of all the fiddly details for you, allowing you to concentrate on the data you wish to exchange.

Basics

Messages are broken up into two types of frame: an address frame, where the master indicates the slave to which the message is being sent, and one or more data frames, which are 8-bit data messages passed from master to slave or vice versa. Data is placed on the SDA line after SCL goes low, and is sampled after the SCL line goes high. The time between clock edge and data read/write is defined by the devices on the bus and will vary from chip to chip.

Start Condition

To initiate the address frame, the master device leaves SCL high and pulls SDA low. This puts all slave devices on notice that a transmission is about to start. If two master devices wish to take ownership of the bus at one time, whichever device pulls SDA low first wins the race and gains control of the bus. It is possible to issue repeated starts, initiating a new communication sequence without relinquishing control of the bus to other masters; we'll talk about that later.

Stop condition

Once all the data frames have been sent, the master will generate a stop condition. Stop conditions are defined by a 0->1 (low to high) transition on SDA after a 0->1 transition on SCL, with SCL remaining high. During normal data writing operation, the value on SDA should **not** change when SCL is high, to avoid false stop conditions.

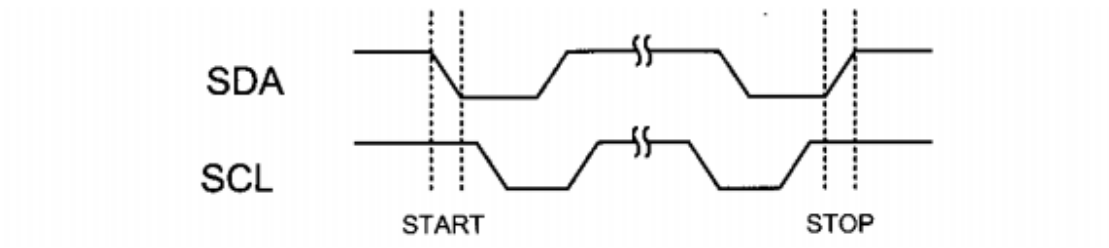


Figure 11: Start & Stop Condition

Address Frame

The address frame is always first in any new communication sequence. For a 7-bit address, the address is clocked out most significant bit (MSB) first, followed by a R/W bit indicating whether this is a read (1) or write (0) operation.
The 9th bit of the frame is the NACK/ACK bit. This is the case for all frames (data or address).
Once the first 8 bits of the frame are sent, the receiving device is given control over SDA. If the receiving device does not pull the SDA line low before the 9th clock pulse, it can be inferred that the receiving device either did not receive the data or did not know how to parse the message. In that case, the exchange halts.

Data Frames

After the address frame has been sent, data can begin being transmitted. The master will simply continue generating clock pulses at a regular interval, and the data will be placed on SDA by either the master or the slave, depending on whether the R/W bit indicated a read or write operation. The number of data frames is arbitrary, and most slave devices will auto-increment the internal register, meaning that subsequent reads or writes will come from the next register in line.

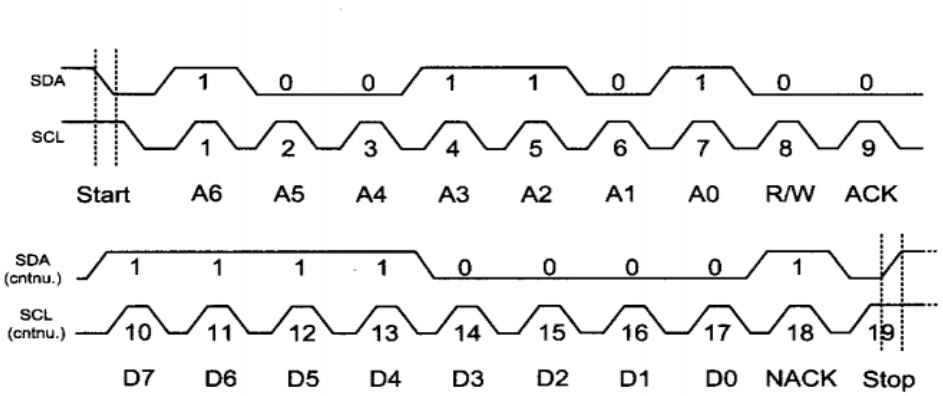


Figure 12: Address Frame

10-bit Addresses

In a 10-bit addressing system, two frames are required to transmit the slave address. The first frame will consist of the code b11110xyz, where ‘x’ is the MSB of the slave address, y is bit 8 of the slave address, and z is the read/write bit as described above. The first frame’s ACK bit will be asserted by all slaves which match the first two bits of the address. As with a normal 7-bit transfer, another transfer begins immediately, and this transfer contains bits 7:0 of the address. At this point, the addressed slave should respond with an ACK bit. If it doesn’t, the failure mode is the same as a 7-bit system.
Note that 10-bit address devices can coexist with 7-bit address devices, since the

leading ‘11110’ part of the address is not a part of any valid 7-bit addresses.

Repeated Start Conditions

Sometimes, it is important that a master device be allowed to exchange several messages in one go, without allowing other master devices on the bus to interfere. For this reason, the repeated start condition has been defined.
To perform a repeated start, SDA is allowed to go high while SCL is low, SCL is allowed to go high, and then SDA is brought low again while SCL is high. Because there was no stop condition on the bus, the previous communication wasn’t truly completed and the current master maintains control of the bus.
At this point, the next message can begin transmission. The syntax of this new message is the same as any other message–an address frame followed by data frames. Any number of repeated starts is allowed, and the master will maintain control of the bus until it issues a stop condition.

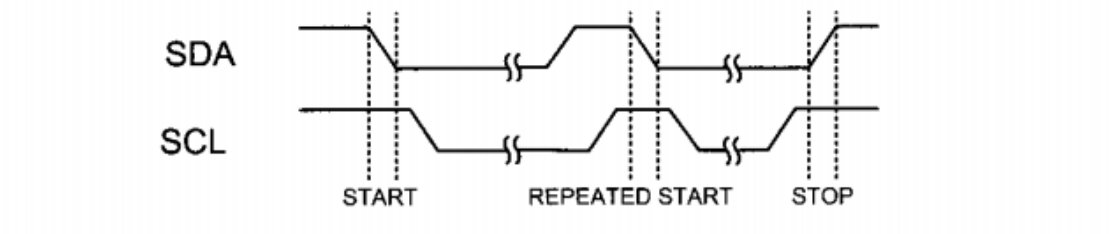


Figure 13: Repeated Start Condition

2.4 Motors Control

2.4.1 H bridge

An **H bridge** is an electronic circuit that enables a voltage to be applied across a load in opposite direction. These circuits are often used in robotics and other applications to allow DC motors to run forwards or backwards.

Most DC-to-AC converters (power inverters), most AC/AC converters, the DC-to-DC push–pull converter, most motor controllers, and many other kinds of power electronics use H bridges. In particular, a bipolar stepper motor is almost invariably driven by a motor controller containing two H-bridges.

2.4.2 General concept

H bridges are available as integrated circuits, or can be built from discrete components.

The term H bridge is derived from the typical graphical representation of such a circuit. An H bridge is built with four switches (solid-state or mechanical). When the switches S1 and S4 (according to the first figure) are closed (and S2 and S3 are open) a positive voltage will be applied across the motor. By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

Using the nomenclature above, the switches S1 and S2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches S3 and S4. This condition is known as shoot-through.

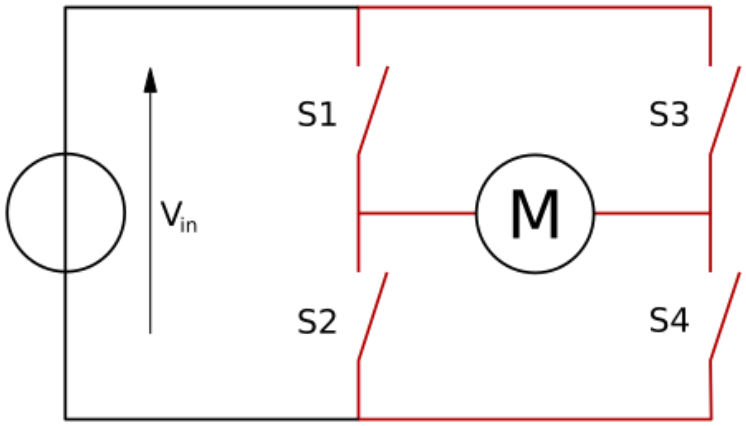


Figure 14: H-Bridge Concept

2.4.3 Operation

The H-bridge arrangement is generally used to reverse the polarity/direction of the motor, but can also be used to 'brake' the motor, where the motor comes to a sudden stop, as the motor's terminals are shorted, or to let the motor 'free run' to a stop, as the motor is effectively disconnected from the circuit.

2.4.4 Construction

Relays

One way to build an H bridge is to use an array of relays from a relay board.

A "Double Pole Double Throw" (DPDT) relay can generally achieve the same electrical functionality as an H bridge (considering the usual function of the device). However a semiconductor-based H bridge would be preferable to the relay where a smaller physical size, high speed switching, or low driving voltage (or low driving power) is needed, or where the wearing out of mechanical parts is undesirable.

Another option is to have a DPDT relay to set the direction of current flow and a transistor to enable the current flow. This can extend the relay life, as the relay will be

switched while the transistor is off and thereby there is no current flow. It also enables the use of PWM switching to control the current level.

N and P channel semiconductors

A solid-state H bridge is typically constructed using opposite polarity devices, such as PNP bipolar junction transistors (BJT) or P-channel MOSFETs connected to the high voltage bus and NPN BJTs or N-channel MOSFETs connected to the low voltage bus.

N channel-only semiconductors

The most efficient MOSFET designs use N-channel MOSFETs on both the high side and low side because they typically have a third of the ON resistance of P-channel MOSFETs. This requires a more complex design since the gates of the high side MOSFETs must be driven positive with respect to the DC supply rail. Many integrated circuit MOSFET gate drivers include a charge pump within the device to achieve this.

Alternatively, a switched-mode power supply DC–DC converter can be used to provide isolated ('floating') supplies to the gate drive circuitry. A multiple-output fly back converter is well-suited to this application.

Another method for driving MOSFET-bridges is the use of a specialized transformer known as a GDT (Gate Drive Transformer), which gives the isolated outputs for driving the upper FETs gates. The transformer core is usually a ferrite toroid, with 1:1 or 4:9 winding ratio. However, this method can only be used with high frequency signals. The design of the transformer is also very important, as the leakage inductance should be minimized, or cross conduction may occur. The outputs of the transformer are usually clamped by Zener diodes, because high voltage spikes could destroy the MOSFET gates.

Variants

A common variation of this circuit uses just the two transistors on one side of the load, similar to a class AB amplifier. Such a configuration is called a "half bridge". The half bridge is used in some switched-mode power supplies that use synchronous rectifiers and in switching amplifiers. The half-H bridge type is commonly abbreviated to "Half-H" to distinguish it from full ("Full-H") H bridges. Another common variation, adding a third 'leg' to the bridge, creates a three-phase inverter. The three-phase inverter is the core of any AC motor drive.

A further variation is the half-controlled bridge, where the low-side switching device on one side of the bridge, and the high-side switching device on the opposite side of the bridge, are each replaced with diodes. This eliminates the shoot-through failure

mode, and is commonly used to drive variable or switched reluctance machines and actuators where bi-directional current flow is not required.

2.4.5 Operation as an inverter

A common use of the H-bridge is an inverter. The arrangement is sometimes known as a single-phase bridge inverter.

The H-bridge with a DC supply will generate a square wave voltage waveform across the load. For a purely inductive load, the current waveform would be a triangle wave, with its peak depending on the inductance, switching frequency, and input voltage.

2.4.6 Motor Driver

A **motor driver** is a little current amplifier; the function of **motor drivers** is to take a low-current control signal and then turn it into a higher-current signal that can **drive** a **motor**.

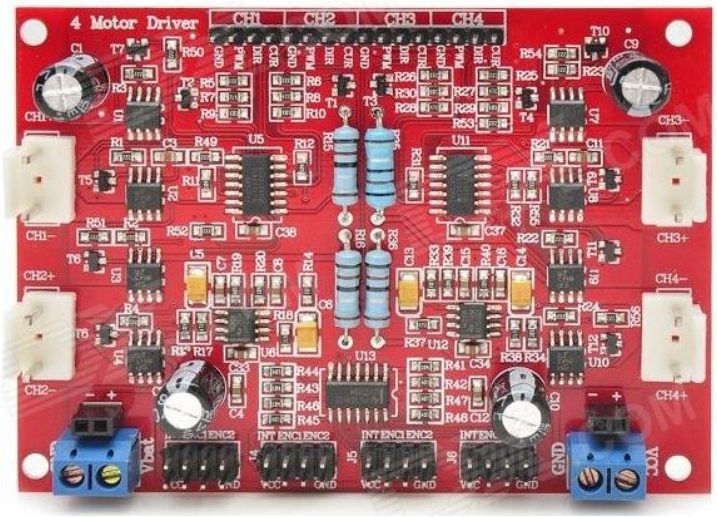


Figure 15: Motor Driver Board

The one used in this project has 4 channels that can output up to 2 ampere current.

2.4.7 What is Pulse-width Modulation?

Pulse Width Modulation (PWM) is a fancy term for describing a type of digital signal. Pulse width modulation is used in a variety of applications including sophisticated control circuitry. It's possible to accomplish a range of results in both applications because pulse width modulation allows us to vary how much time the signal is high in an analog fashion. While the signal can only be high (usually 5V) or low (ground) at any time, we can change the proportion of time the signal is high compared to when it is low over a consistent time interval.

2.4.8 Duty Cycle

When the signal is high, we call this “on time”. To describe the amount of “on time” , we use the concept of duty cycle. Duty cycle is measured in percentage. The percentage duty cycle specifically describes the percentage of time a digital signal is on over an interval or period of time. This period is the inverse of the frequency of the waveform.

If a digital signal spends half of the time on and the other half off, we would say the digital signal has a duty cycle of 50% and resembles an ideal square wave. If the percentage is higher than 50%, the digital signal spends more time in the high state than the low state and vice versa if the duty cycle is less than 50%. Here is a graph that illustrates these three scenarios:

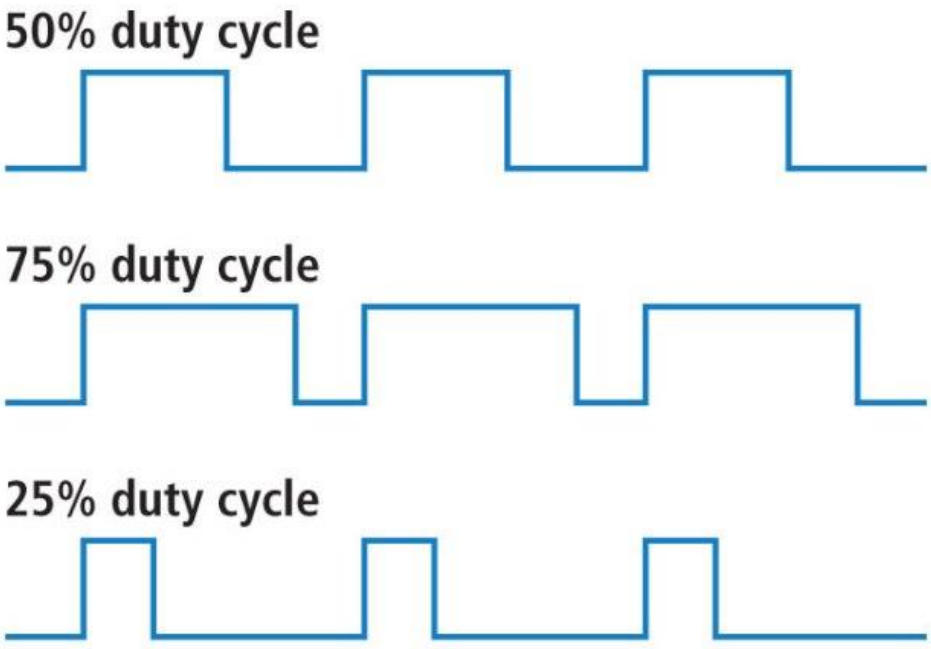


Figure 16: PWM Duty Cycles

100% duty cycle would be the same as setting the voltage to 5 Volts (high). 0% duty cycle would be the same as grounding the signal.

CHAPTER 3:

Object Recognition

3.1 Machine Learning

3.1.1 What is machine learning?

Machine learning is one of the best field of Computer science. Machine learning is a type of artificial intelligence (AI) that provides computers with the ability to learn without being explicitly programmed. Machine learning focuses on the development of computer programs that can teach themselves to grow and change when exposed to new data. Machine learning studies computer algorithms for learning to do stuff. We might, for instance, be interested in learning to complete a task, or to make accurate predictions, or to behave intelligently. The learning that is being done is always based on some sort of observations or data, such as examples (the most common case in this course), direct experience, or instruction. So in general, machine learning is about learning to do better in the future based on what was experienced in the past. The emphasis of machine learning is on automatic methods. In other words, the goal is to devise learning algorithms that do the learning automatically without human intervention or assistance. The machine learning paradigm can be viewed as "programming by example." Often we have a specific task in mind, such as spam filtering. But rather than program the computer to solve the task directly, in machine learning, we seek methods by which the computer will come up with its own program based on examples that we provide. Machine learning is a core subarea of artificial intelligence. It is very unlikely that we will be able to build any kind of intelligent system capable of any of the facilities that we associate with intelligence, such as language or vision, without using learning to get there. These tasks are otherwise simply too difficult to solve. Further, we would not consider a system to be truly intelligent if it were incapable of learning since learning is at the core of intelligence. Although a subarea of AI, machine learning also intersects broadly with other fields, especially statistics, but also mathematics, physics, theoretical computer science and more.

Because of new computing technologies, machine learning today is not like machine learning of the past. It was born from pattern recognition and the theory that computers can learn without being programmed to perform specific tasks; researchers

interested in artificial intelligence wanted to see if computers could learn from data. The iterative aspect of machine learning is important because as models are exposed to new data, they are able to independently adapt. They learn from previous computations to produce reliable, repeatable decisions and results. It's a science that's not new – but one that's gaining fresh momentum.

While many machine learning algorithms have been around for a long time, the ability to automatically apply complex mathematical calculations to big data – over and over, faster and faster – is a recent development.

3.1.2 Types of Machine learning algorithms

There some variations of how to define the types of Machine Learning Algorithms but commonly they can be divided into categories according to their purpose and the main categories are the following:

- **Supervised learning**
- **Unsupervised Learning**
- **Semi-supervised Learning**
- **Reinforcement Learning**

3.1.2.1 Supervised Learning

I like to think of supervised learning with the concept of function approximation, where basically we train an algorithm and in the end of the process we pick the function that best describes the input
data, the one that for a given X makes the best estimation of y (X -> y). Most of the time we are not able to figure out the true function that always make the correct predictions and other reason is that the algorithm rely upon an assumption made by humans about how the computer should learn and this
assumptions introduce a bias, Bias is topic I'll explain in another post.

Here the human Experts acts as the teacher where we feed the computer with training data containing the input/predictors and we show it the correct answers (output) and from the data the computer should be able to learn the patterns.

Supervised learning algorithms try to *model relationships and dependencies between the target prediction output and the input features* such that we can predict the output values for new data based on those relationships which it learned from the previous data sets.

3.1.2.2 Unsupervised Learning

The computer is trained with unlabeled data.

Here there's no teacher at all, actually the computer might be able to teach you new things after it learns patterns in data, these algorithms a particularly useful in cases where the human expert doesn't know what to look for in the data.

are the family of machine learning algorithms which are mainly used in *pattern detection* and *descriptive modeling*. However, *there are no output categories or labels* here based on which the algorithm can try to model relationships. These algorithms try to use techniques on the input data to *mine for rules*, *detect patterns*, and *summarize and group the data points* which help in deriving meaningful insights and describe the data better to the users.

3.1.2.3 Semi-supervised Learning

In the previous two types, either there are no labels for all the observation in the dataset or labels are present for all the observations. Semi-supervised learning falls in between these two. In many practical situations, the cost to label is quite high, since it requires skilled human experts to do that. So, in the absence of labels in the majority of the observations but present in few, semi-supervised algorithms are the best candidates for the model building. These methods exploit the idea that even though the group memberships of the unlabeled data are unknown, this data carries important information about the group parameters.

3.1.2.4 Reinforcement Learning

Method aims at using observations gathered from the interaction with the environment to take actions that would maximize the reward or minimize the risk. Reinforcement learning algorithm (called the agent) continuously learns from the environment in an iterative fashion. In the process, the agent learns from its experiences of the environment until it explores the full range of possible states.

Reinforcement Learning is a type of Machine Learning, and thereby also a branch of Artificial Intelligence. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance. Simple reward feedback is required for the agent to learn its behavior; this is known as the reinforcement signal.

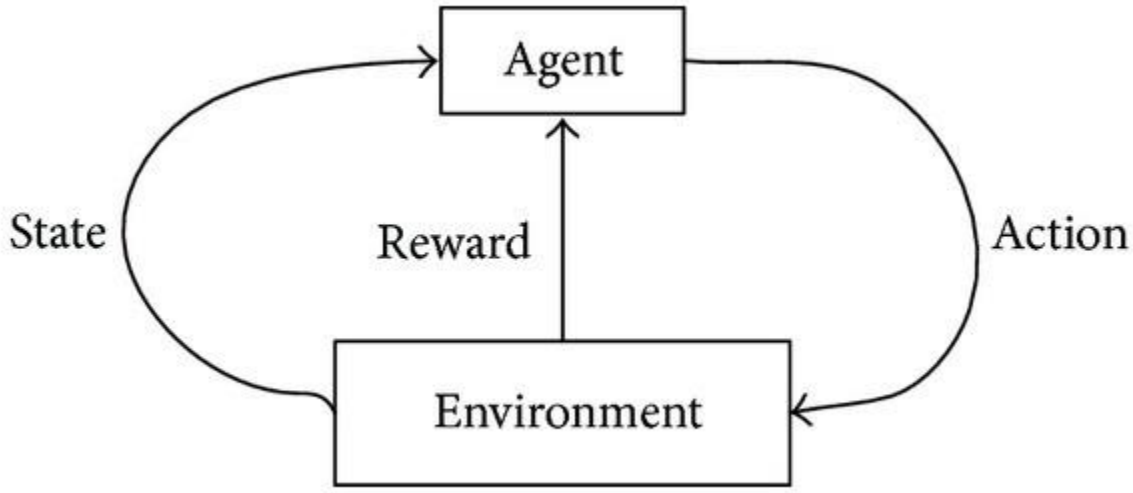


Figure 17: Reinforcement Learning Algorithm

There are many different algorithms that tackle this issue. As a matter of fact, Reinforcement Learning is defined by a specific type of problem, and all its solutions are classed as Reinforcement Learning algorithms. In the problem, an agent is supposed decide the best action to select based on his current state. When this step is repeated, the problem is known as a Markov Decision Process.

In order to produce intelligent programs (also called agents), reinforcement learning goes through the following steps:

Input state is observed by the agent.

Decision making function is used to make the agent perform an action.

After the action is performed, the agent receives reward or reinforcement from the environment.

The state-action pair information about the reward is stored.

3.1.3 Common machine learning algorithms

Random Forest

Decision trees use directed graphs to model decision making; each node on the graph represents a question about the data (“Is income greater than $70,000?”) and the branches stemming from each node represent the possible answers to that question.

Compounding hundreds or even thousands of these decision trees is an “ensemble” method called a random forest.

Though highly accurate, random forests are often dubbed black box models because they are complex to the point that they can be difficult to interpret. For example, understanding how a random forest model approves or denies a loan could involve sifting through thousands of finely-tuned decisions. Nevertheless, random forest models are popular due to their high accuracy and relatively low computational expense. They are used for a wide variety of applications including churn modeling and customer segmentation.

Logistic Regression

Logistic regression, which is borrowed from the field of classical statistics, is one of the simpler machine learning algorithms. This machine learning technique is commonly used for binary classification problems, meaning those in which there are two possible outcomes that are influenced by one or more explanatory variables. The algorithm estimates the probability of an outcome given a set of observed variables.

Where logistic regression differs from other methods is in its interpretability. Since this algorithm is derived from the highly interpretable linear regression algorithm, the influence of each data feature can be interpreted without much effort. As a result, logistic regression is often favored when interpretability and inference is paramount. This versatile algorithm is used to determine the outcome of binary events such as customer churn, marketing click-throughs, or fraud detection.

Kernel Methods

Kernel methods are a group of machine learning algorithms used for pattern analysis, which involves organizing raw data into rankings, clusters, or classifications. These methods allow data scientists to apply their domain knowledge of a given problem by building custom kernels that incorporate the data transformations that are most likely to improve the accuracy of the overall mode The most popular application of kernels is the support vector machine (SVM), which builds a model that classifies new data as belonging to one category or another based on a set of training examples. A SVM makes these determinations by representing each example as a point in a multi-dimensional space called a hyperplane. The points are then separated into categories by maximizing the distance (called a “margin”) between the different apparent groups in the data.

Kernel methods are useful you have domain knowledge pertaining to the decision boundaries beforehand, which usually isn't true except for the most common problems. As a result, practitioners usually opt for a more “out-of-the-box” machine learning algorithm.

K-Means Clustering

Clustering is a type of unsupervised learning, which is used when working with data that does not have defined categories or groups (unlabeled data). The goal of k-means clustering is to find distinct groups in the data based on inherent similarities between them rather than predetermined labels. K represents the total number of unique groups the algorithm will create. Each example is assigned to one group or another based on similarity to other examples across a set of characteristics called features. K-means clustering is useful for business applications like customer segmentation, inventory categorization, and anomaly detection.

Ultimately, the best machine learning algorithm to use for any given project depends on the data available, how the results will be used, and the data scientist's domain expertise on the subject. Understanding how they differ is a key step to ensuring that every predictive model your data scientists build and deploy delivers valuable results.

Neural Networks

The goal of artificial neural network machine learning algorithms is to mimic the way the human brain organizes and understands information in order to arrive at various predictions. In artificial neural networks, information is passed through an input layer, a hidden layer, and an output layer. The input and output layers can be comprised of raw features and predictions, respectively. The hidden layer in between consists of many highly interconnected neurons capable of complex meta-feature engineering. As the neural network “learns” the data, the connections between these neurons are fine-tuned until the network yields highly accurate predictions.

This biological approach to computation allows neural networks to excel at some of the most challenging, high-dimensional problems in artificial intelligence, such as speech and object recognition, image segmentation, and natural language processing. Like random forests, neural networks are difficult — if not impossible — to interpret without the use of tools like Skater, an open source model interpretation package. This means that data scientists will often defer to simpler machine learning algorithms unless their analysis demands superior accuracy.

For our purpose of object recognition, we are only interested in neural networks, so we are going to discuss it with more details next.

3.2 Neural Networks

3.2.1 Definition

The definition of a neural network, more properly referred to as an 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Or you can also think of Artificial Neural Network as computational model that is inspired by the way biological neural networks in the human brain process information.

3.2.2 Biological motivation and connections

The basic computational unit of the brain is a **neuron**. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14}—10^{15} **synapses**. The diagram below shows a cartoon drawing of a biological neuron (left) and a common mathematical model (right).

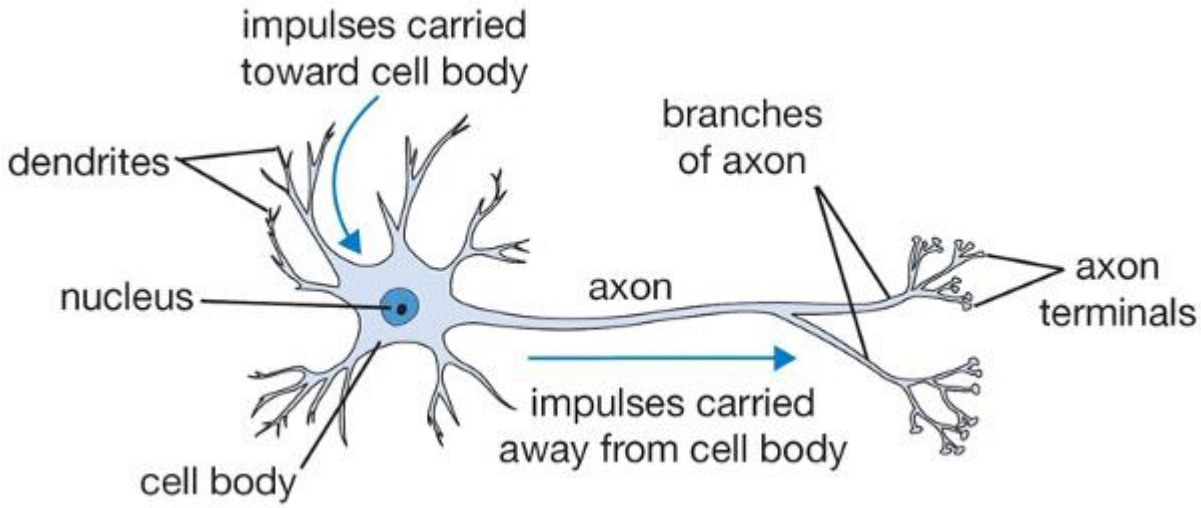


Figure 18: Human Neuron

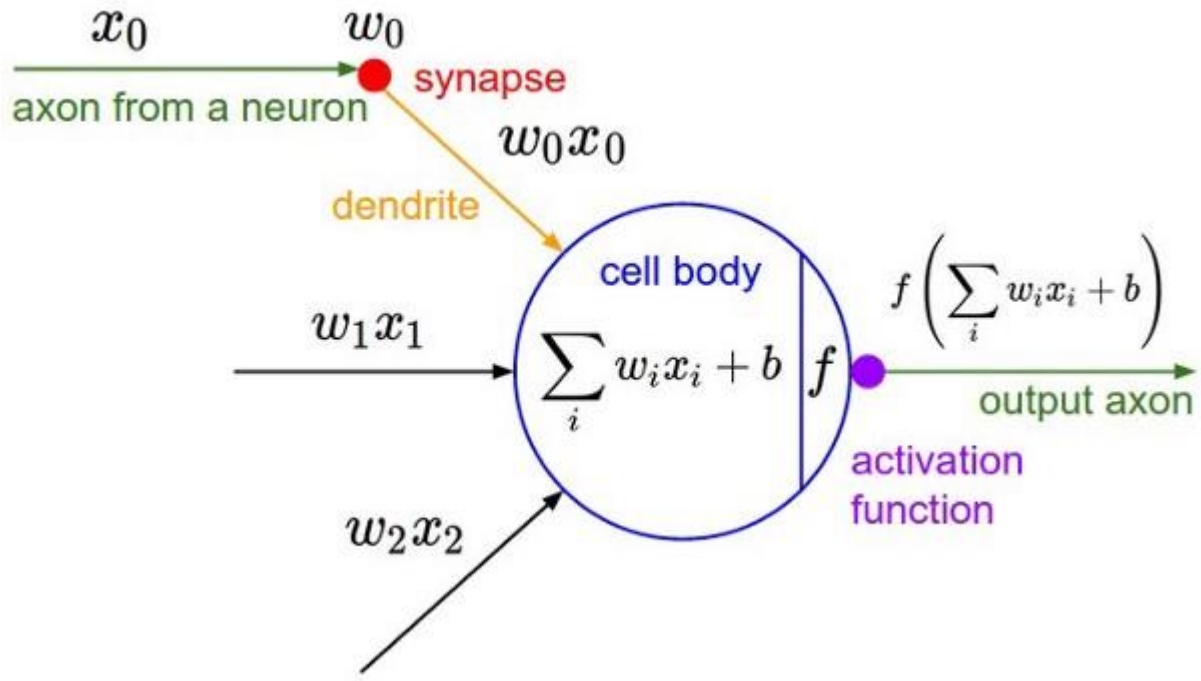


Figure 19: Artificial Neuron

Biological neuron (left) and a common mathematical model (right)

The basic unit of computation in a neural network is the neuron, often called a node or unit. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated
weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function to the weighted sum of its inputs.

The idea is that the synaptic strengths (the weights *w*) are learnable and control the strength of influence and its direction: excitatory (positive weight) or inhibitory (negative weight) of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can *fire*, sending a spike along its axon. In the computational model, we assume that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. We model the *firing rate* of the neuron with an **activation function** *(e.x sigmoid function)*, which represents the frequency of the spikes along the axon.

Artificial neural networks doesn't work like our brain, ANN are simple crude comparison, the connections between biological networks are much more complex than those implemented by **Artificial neural network** architectures, remember, our brain is much more complex and there is more we need to learn from it. There are

many things we don't know about our brain and this also makes hard to know how we should model an Artificial Brain to reason at human level. Whenever we train a neural network, we want our model to learn the optimal weights (w) that best predicts the desired outcome (y) given the input signals or information (x).

3.2.3 Neural Network Architecture

From the above explanation we can conclude that a neural network is made of neurons, biologically the neurons are connected through synapses where information flows (weights for out computational model), when we train a neural network we want the neurons to fire whenever they learn specific patterns from the data, and we model the fire rate using an activation function.

But that's not everything…

Input Nodes (input layer): No computation is done here within this layer, they just pass the information to the next layer (hidden layer most of the time). A block of nodes is also called **layer**.

Hidden nodes (hidden layer): In Hidden layers is where intermediate processing or computation is done, they perform computations and then transfer the weights (signals or information) from the input layer to the following layer (another hidden layer or to the output layer). It is possible to have a neural network without a hidden layer and I'll come later to explain this.

Output Nodes (output layer): Here we finally use an activation function that maps to the desired output format (e.g. Softmax for classification).

Connections and weights: The *network* consists of connections, each connection transferring the output of a neuron i to the input of a neuron *j*. In this sense *i* is the predecessor of *j* and *j* is the successor of *i*, each connection is assigned a weight *Wij.*

Activation function: the **activation function** of a node defines the output of that node given an input or set of inputs. A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, it is the *nonlinear* activation function that allows such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

Learning rule: The *learning rule* is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This *learning* process typically amounts to modifying the weights and thresholds.

3.2.4 Types of Neural Networks

There are many classes of neural networks and these classes also have sub-classes, here I will list the most used ones and make things simple to move on in this journey to learn neural networks.

3.2.4.1 Feedforward Neural Network

A feedforward neural network is an artificial neural network where connections between the units do *not* form a cycle. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

We can distinguish two types of feedforward neural networks:

Single-layer Perceptron

This is the simplest feedforward neural Network and does not contain any hidden layer, which means it only consists of a single layer of output nodes. This is said to be single because when we count the layers we do not include the input layer, the reason for that is because at the input layer no computations is done, the inputs are fed directly to the outputs via a series of weights.

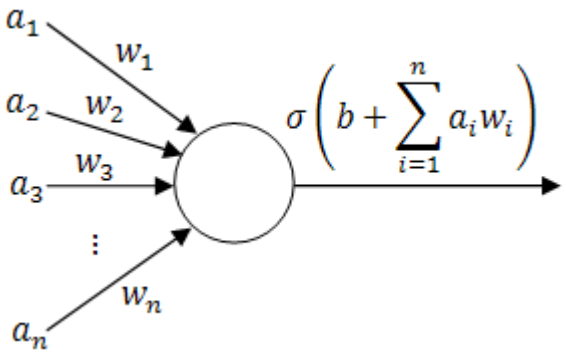


Figure 20: Single Layer Perceptron

Multi-layer perceptron (MLP)

This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function. MLP are very more useful and one good reason is that, they are able to learn non-linear representations (most of the cases the data presented to us is not linearly separable).

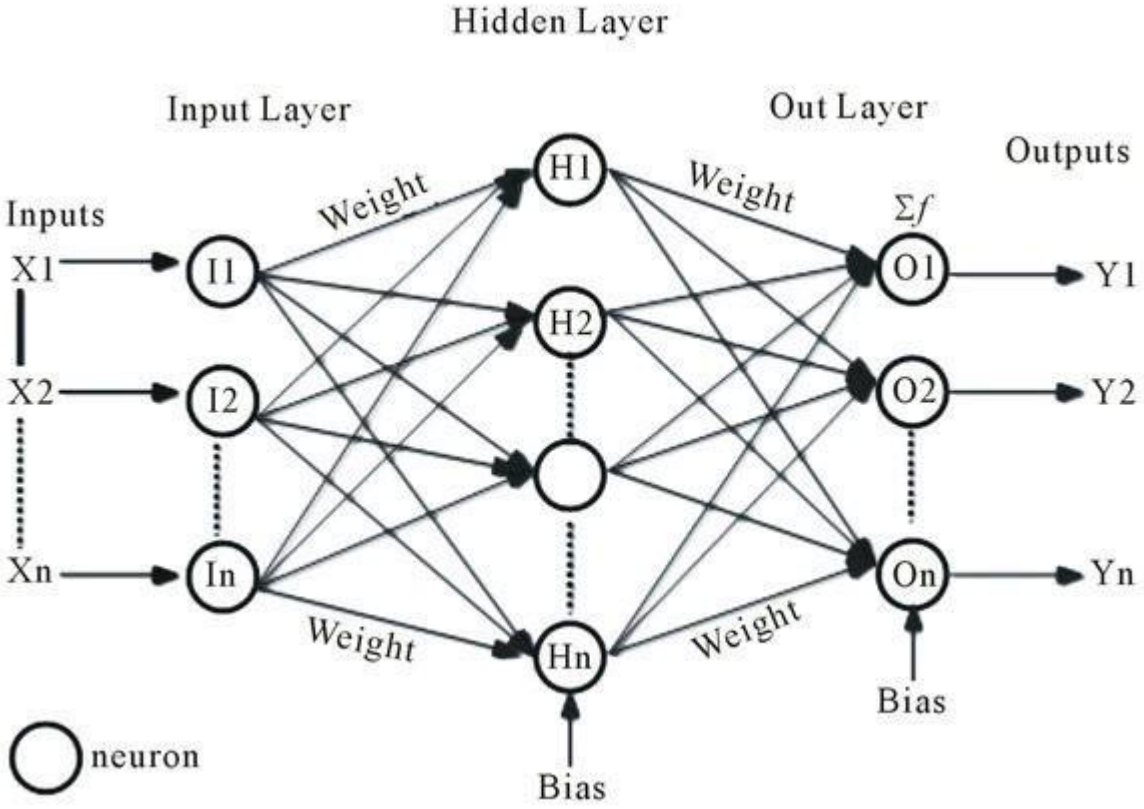


Figure 21: MLP

3.2.4.2 Convolutional Neural Network (CNN)

Convolutional Neural Networks are very similar to ordinary Neural Networks, they are made up of neurons that have learnable weights and biases. In convolutional neural network (CNN, or ConvNet or shift invariant or space invariant) the unit connectivity pattern is inspired by the organization of the visual cortex, Units respond to stimuli in a restricted region of space known as the receptive field. Receptive fields partially overlap, over-covering the entire visual field. Unit response can be approximated mathematically by a convolution operation. They are variations of multilayer perceptrons that use minimal preprocessing. Their wide applications is in

image and video recognition, recommender systems and natural language processing. CNNs requires large data to train on.

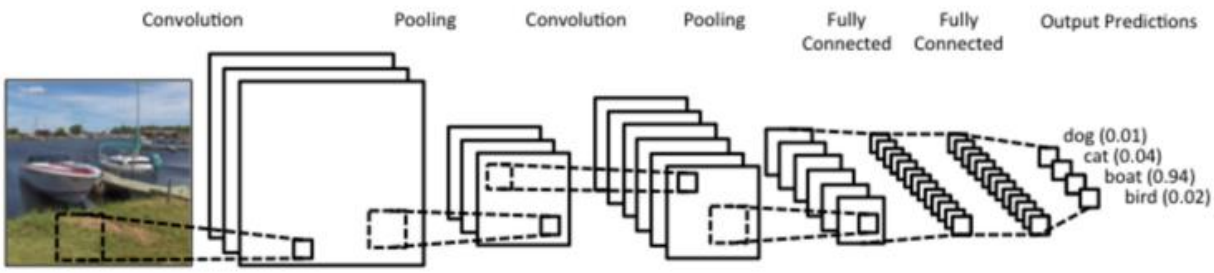


Figure 22: CNN for Image Classification

3.2.4.3 Recurrent neural networks

In recurrent neural network (RNN), connections between units form a directed cycle (they propagate data forward, but also backwards, from later processing stages to earlier stages). This allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and other general sequence processors.

3.3 CONVOLUTIONAL NEURAL NETWORKS

3.3.1 What are Convolutional Neural Network?

Convolutional Neural Networks (**ConvNets** or **CNNs**) are one of the main reasons why deep learning is so popular today. They are a very effective class of neural networks that is highly effective at classifying structured data where the order of arrangement matters. Such data include images, audio and video. Their uses span a wide range of domains, however, they are primarily used for image classification, object detection, generative models and image segmentation. As of recently, their performance on image recognition tasks has surpassed human performance on standard datasets.

Figure 23: Explaining Figure

In the explaining **Figure 23** above, a ConvNet is able to recognize scenes and the system is able to suggest relevant captions ("a soccer player is kicking a soccer ball") while **Figure 24** shows an example of ConvNets being used for recognizing everyday objects, humans and animals. Lately, ConvNets have been effective in several Natural Language Processing tasks (such as sentence classification) as well.

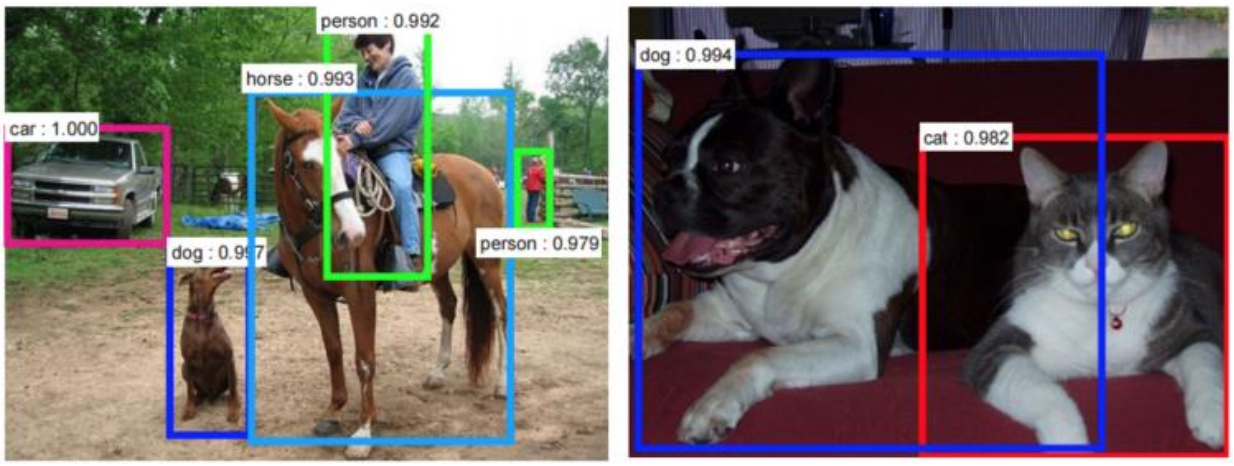


Figure 24: Explaining Figure 2

ConvNets, therefore, are an important tool for most machine learning practitioners today. However, understanding ConvNets and learning to use them for the first time can sometimes be an intimidating experience. The primary purpose of this blog post is to develop an understanding of how Convolutional Neural Networks work on images.

3.3.2 The LeNet Architecture (1990s)

LeNet was one of the very first convolutional neural networks which helped propel the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since the year 1988. At that time the LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits, etc.

Below, we will develop an intuition of how the LeNet architecture learns to recognize images. There have been several new architectures proposed in the recent years which are improvements over the LeNet, but they all use the main concepts from the LeNet and are relatively easier to understand if you have a clear understanding of the former.

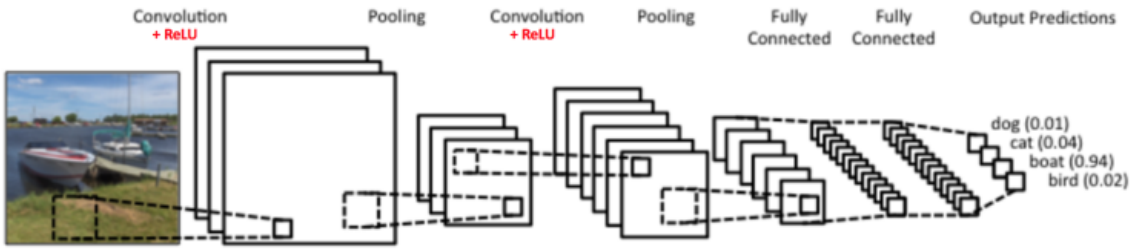


Figure 25: The Convolutional Neural Network

The Convolutional Neural Network in **Figure 25** is similar in architecture to the original LeNet and classifies an input image into four categories: dog, cat, boat or bird (the original LeNet was used mainly for character recognition tasks). As evident from the figure above, on receiving a boat image as input, the network correctly assigns the highest probability for boat (0.94) among all four categories. The sum of all probabilities in the output layer should be one (explained later in this post).

There are four main operations in the ConvNet shown in **Figure 25** above:

- Convolution
- Non-Linearity (ReLU)
- Pooling or Sub Sampling
- Classification (Fully Connected Layer)

These operations are the basic building blocks of *every* Convolutional Neural Network, so understanding how this work is an important step to developing a sound understanding of ConvNets. We will try to understand the intuition behind each of these operations below.

3.3.2.1 An Image is a matrix of pixel values

Essentially, every image can be represented as a matrix of pixel values.

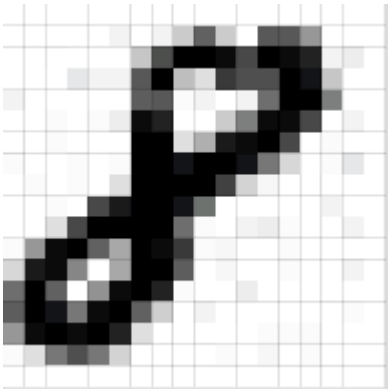


Figure 26: Pixel Image of number "8"

Channel is a conventional term used to refer to a certain component of an image. An image from a standard digital camera will have three channels – red, green and blue – you can imagine those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

A grayscale image, on the other hand, has just one channel. For the purpose of this post, we will only consider grayscale images, so we will have a single 2d matrix representing an image. The value of each pixel in the matrix will range from 0 to 255 – zero indicating black and 255 indicating white.

3.3.2.2 The Convolution Step

ConvNets derive their name from the "convolution" operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. We will not go into the mathematical details of Convolution here, but will try to understand how it works over images.

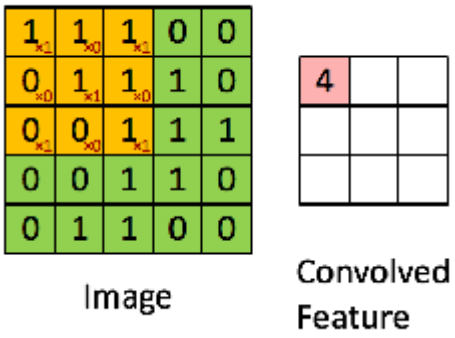
As we discussed above, every image can be considered as a matrix of pixel values. Consider a 5 x 5 image whose pixel values are only 0 and 1 (note that for a grayscale image, pixel values range from 0 to 255, the green matrix below is a special case where pixel values are only 0 and 1):

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Also, consider another 3 x 3 matrix as shown below:

1	0	1
0	1	0
1	0	1

Then, the Convolution of the 5 x 5 image and the 3 x 3 matrix can be computed.



Take a moment to understand how the computation above is being done. We slide the orange matrix over our original image (green) by 1 pixel (also called 'stride') and for every position, we compute element wise multiplication (between the two matrices) and add the multiplication outputs to get the final integer which forms a single element of the output matrix (pink). Note that the 3×3 matrix "sees" only a part of the input image in each stride.

In CNN terminology, the 3×3 matrix is called a '**filter**' or 'kernel' or 'feature detector' and the matrix formed by sliding the filter over the image and computing the dot product is called the 'Convolved Feature' or 'Activation Map' or the '**Feature Map**'. It is important to note that filters acts as feature detectors from the original input image.

It is evident from the animation above that different values of the filter matrix will produce different Feature Maps for the same input image. As an example, consider the following input image:

Figure 27: Input Image

In the table below, we can see the effects of convolution of the above image with different filters. As shown, we can perform operations such as Edge Detection, Sharpen and Blur just by changing the numeric values of our filter matrix before the convolution operation.

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 28: Filter to the Input

Another good way to understand the Convolution operation is by looking at the animation in **Figure 29** below:

Figure 29: The Convolution Operation

A filter (with red outline) slides over the input image (convolution operation) to produce a feature map. The convolution of another filter (with the green outline), over the same image gives a different feature map as shown. It is important to note that the Convolution operation captures the local dependencies in the original image. Also notice how these two different filters generate different feature maps from the same original image. Remember that the image and the two filters above are just numeric matrices as we have discussed above.

In practice, a CNN *learns* the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the network before the training process). The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

The size of the Feature Map (Convolved Feature) is controlled by three parameters that we need to decide before the convolution step is performed:

Depth: Depth corresponds to the number of filters we use for the convolution operation. In the network shown in **Figure 30**, we are performing convolution of the original boat image using three distinct filters, thus producing three different feature maps as shown. You can think of these three feature maps as stacked 2d matrices, so, the ‘depth’ of the feature map would be three.

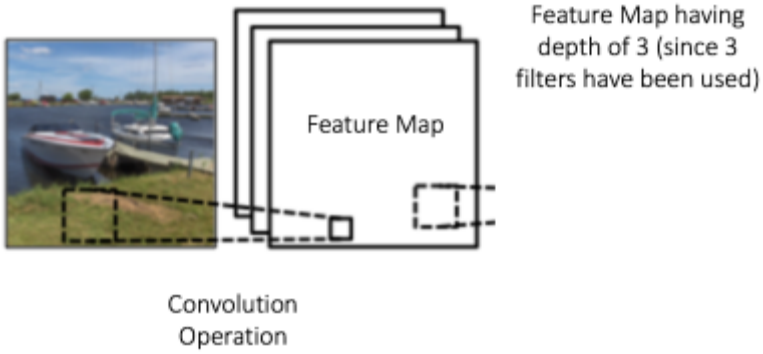


Figure 30: Convolution

Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps.

Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero padding is that it allows us to control the size of the feature maps. Adding zero-padding is also called *wide convolution*, and not using zero-padding would be a *narrow convolution*. This has been explained clearly in [14].

3.3.2.3 Introducing Non-Linearity (ReLU)

An additional operation called ReLU has been used after every Convolution operation in **Figure 31** below. ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

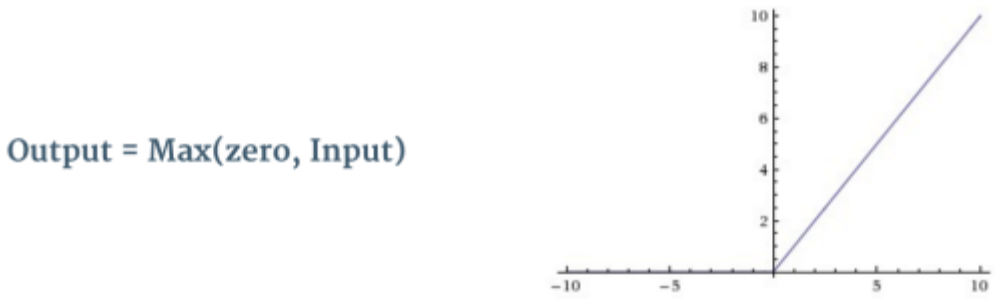


Figure 31: The ReLU operation

ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear operation – element wise matrix

multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

The ReLU operation can be understood clearly from **Figure 32** below. It shows the ReLU operation applied to one of the feature maps obtained in **Figure 32** above. The output feature map here is also referred to as the 'Rectified' feature map.

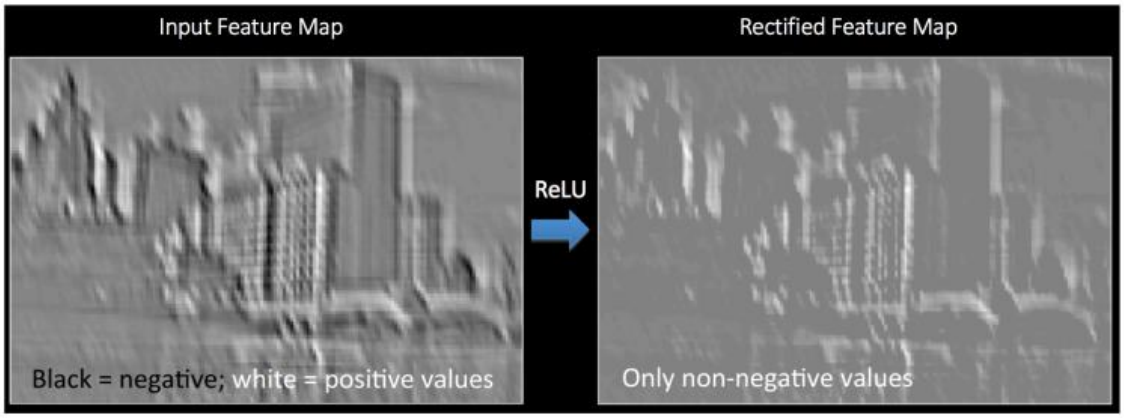


Figure 32: ReLU operation

Other nonlinear functions such as tanh or sigmoid can also be used instead of ReLU, but ReLU has been found to perform better in most situations.

3.3.2.4 The Pooling Step

Spatial Pooling (also called subsampling or down sampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.

Figure 33 shows an example of Max Pooling operation on a Rectified Feature map (obtained after convolution + ReLU operation) by using a 2×2 window.

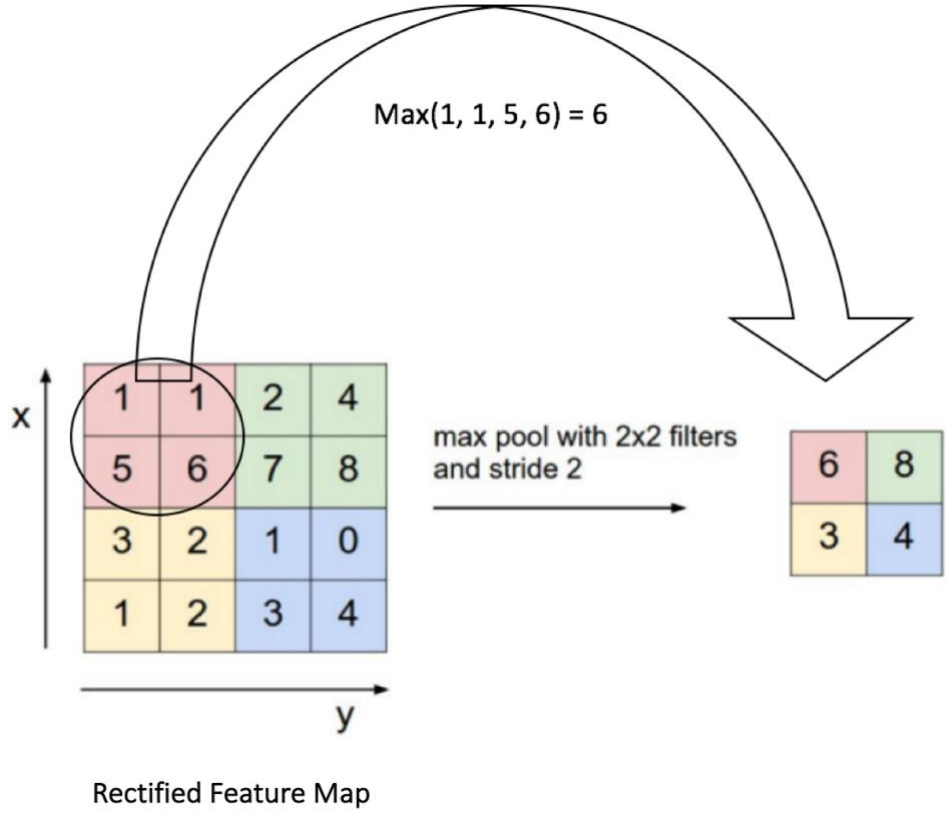


Figure 33: Max Pooling

We slide our 2 x 2 window by 2 cells (also called 'stride') and take the maximum value in each region. As shown in **Figure 33**, this reduces the dimensionality of our feature map.

In the network shown in **Figure 34,** pooling operation is applied separately to each feature map (notice that, due to this, we get three output maps from three input maps).

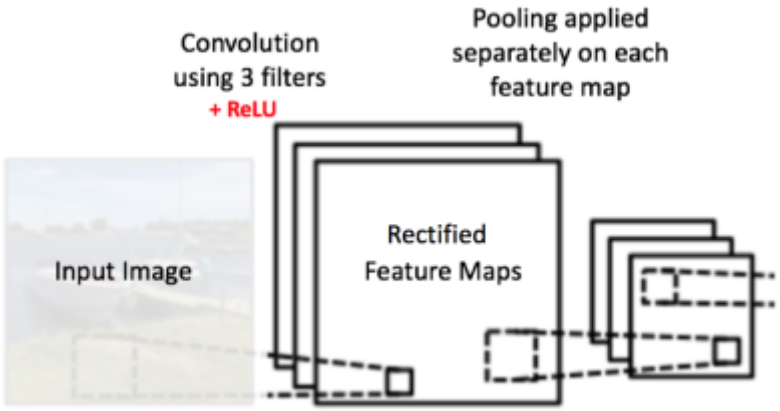


Figure 34: Pooling applied to Rectified Feature Maps

Figure 35 shows the effect of Pooling on the Rectified Feature Map we received after the ReLU operation in **Figure 32** above.

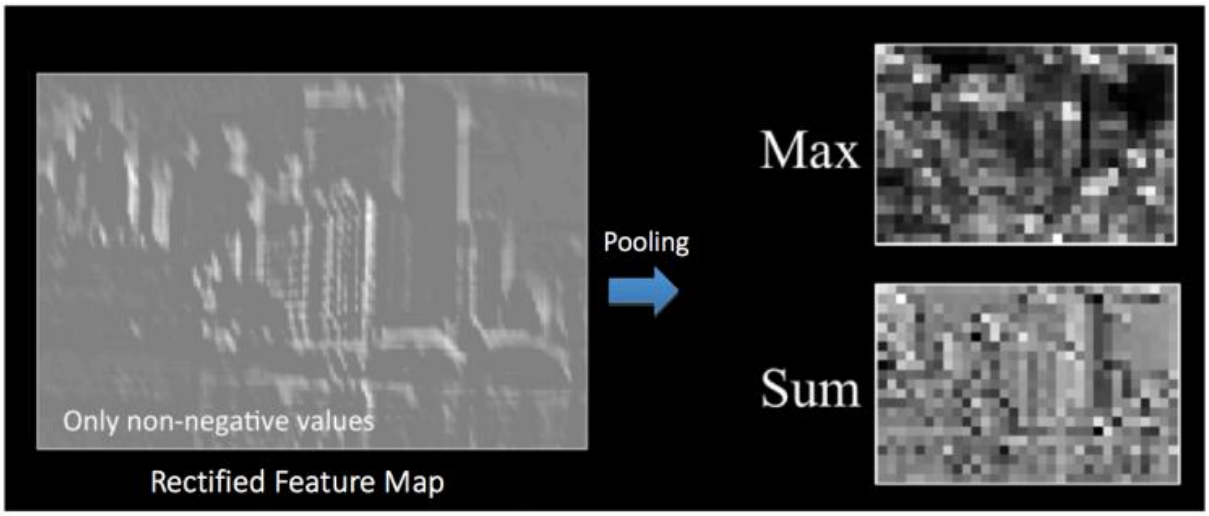


Figure 35: Pooling Source

The function of Pooling is to progressively reduce the spatial size of the input representation. In particular, pooling makes the input representations (feature dimension) smaller and more manageable reduces the number of parameters and computations in the network, therefore, controlling overfitting makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood).

Helps us arrive at an almost scale invariant representation of our image (the exact term is "equivariant"). This is very powerful since we can detect objects in an image no matter where they are located.

Story so far

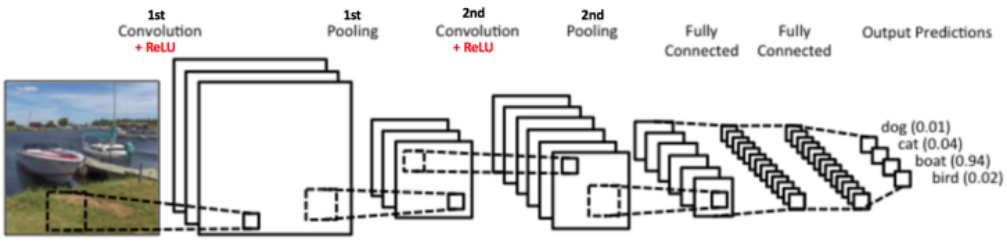


Figure 36: Convolutional Network

So far we have seen how Convolution, ReLU and Pooling work. It is important to understand that these layers are the basic building blocks of any CNN. As shown in **Figure 36**, we have two sets of Convolution, ReLU & Pooling layers – the 2nd

Convolution layer performs convolution on the output of the first Pooling Layer using six filters to produce a total of six feature maps. ReLU is then applied individually on all of these six feature maps. We then perform Max Pooling operation separately on each of the six rectified feature maps.

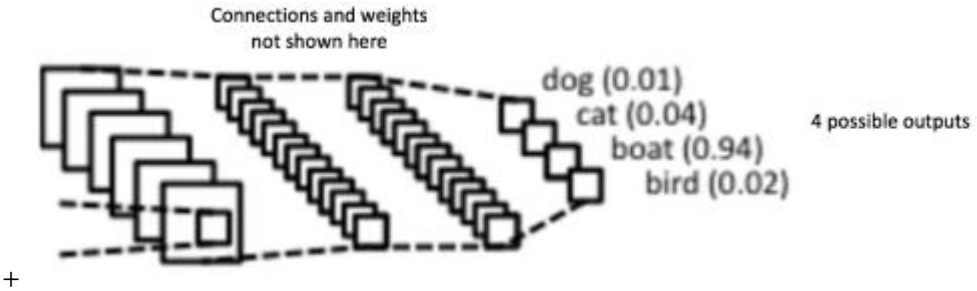
Together these layers extract the useful features from the images, introduce non-linearity in our network and reduce feature dimension while aiming to make the features somewhat equivariant to scale and translation.

The output of the 2nd Pooling Layer acts as an input to the Fully Connected Layer, which we will discuss in the next section.

3.3.2.5 Fully Connected Layer

The Fully Connected layer is a traditional Multi-Layer Perceptron that uses a Softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to Softmax in this post). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer. I recommend reading this post if you are unfamiliar with Multi-Layer Perceptrons.

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. For example, the image classification task we set out to perform has four possible outputs



+

Figure 37: Fully Connected Layer

Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.

he sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one.

3.3.2.6 Putting it all together – Training using Backpropagation

As discussed above, the Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier.

Note that in **Figure 38** below, since the input image is a boat, the target probability is 1 for Boat class and 0 for other three classes, i.e.

Input Image = Boat

Target Vector = [0, 0, 1, 0]

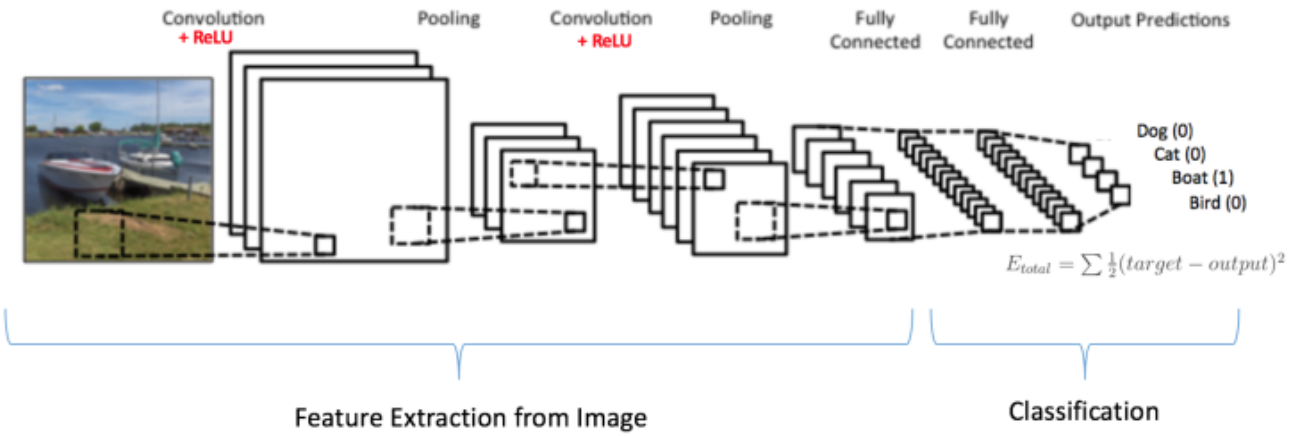


Figure 38: Training the ConvNet

The overall training process of the Convolution Network may be summarized as below:

Step1: We initialize all filters and parameters / weights with random values

Step2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

Let's say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]

Since weights are randomly assigned for the first training example, output probabilities are also random.

Step3: Calculate the total error at the output layer (summation over all 4 classes)

Total Error = ∑ ½ (target probability – output probability) ²

Step4: Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.

The weights are adjusted in proportion to their contribution to the total error.

When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].

This means that the network has *learnt* to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.

Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

Step5: Repeat steps 2-4 with all images in the training set.

The above steps *train* the ConvNet – this essentially means that all the weights and parameters of the ConvNet have now been optimized to correctly classify images from the training set.

When a new (unseen) image is input into the ConvNet, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples). If our training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories.

Note 1: The steps above have been oversimplified and mathematical details have been avoided to provide intuition into the training process. See [4] and [12] for a mathematical formulation and thorough understanding.

Note 2: In the example above we used two sets of alternating Convolution and Pooling layers. Please note however, that these operations can be repeated any number of times in a single ConvNet. In fact, some of the best performing ConvNets today have tens of Convolution and Pooling layers! Also, it is not necessary to have a Pooling layer after every Convolutional Layer. As can be seen in the **Figure 39** below, we can have multiple Convolution + ReLU operations in succession before having a Pooling operation. Also notice how each layer of the ConvNet is visualized in the **Figure 39** below.

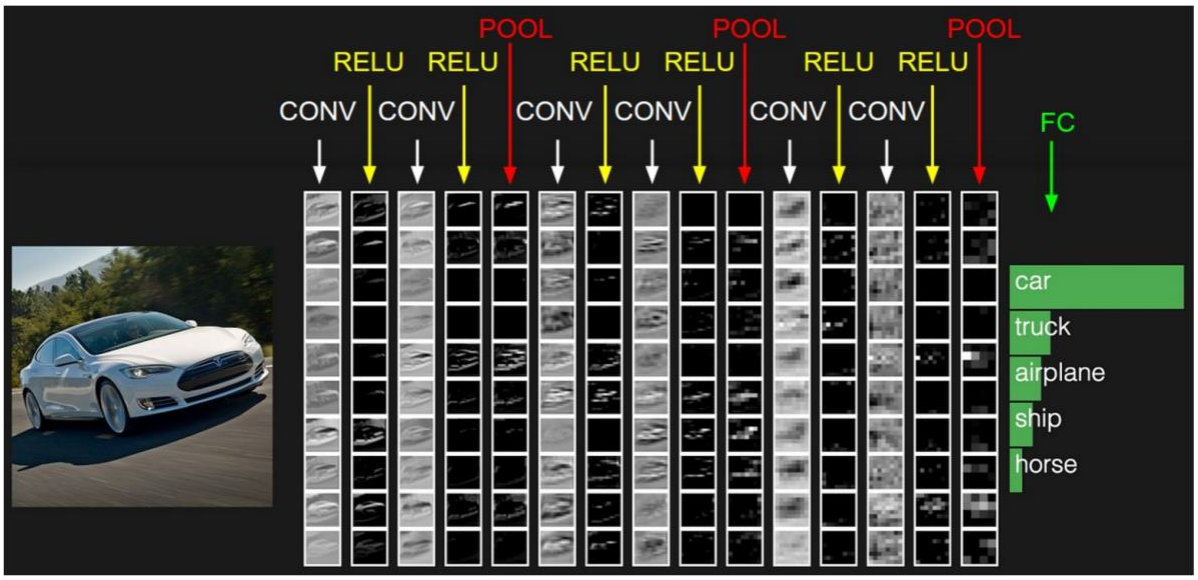


Figure 39: The pooling Layer

3.3.2.7 Visualizing Convolutional Neural Networks

In general, the more convolution steps we have, the more complicated features our network will be able to learn to recognize. For example, in Image Classification a ConvNet may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. This is demonstrated in **Figure 40** below – these features were learnt using a Convolutional Deep Belief Network and the figure is included here just for demonstrating the idea (this is only an example: real life convolution filters may detect objects that have no meaning to humans).

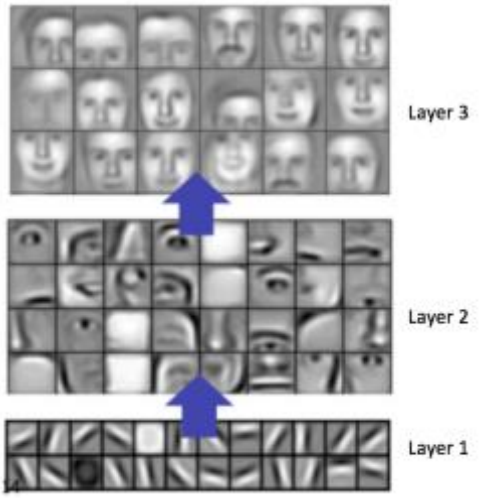


Figure 40: Learned Features from ConvNet

Adam Harley created amazing visualizations of a Convolutional Neural Network trained on the MNIST Database of handwritten digits. I highly recommend playing around with it to understand details of how a CNN works.

We will see below how the network works for an input '8'. Note that the visualization in **Figure 41** does not show the ReLU operation separately.

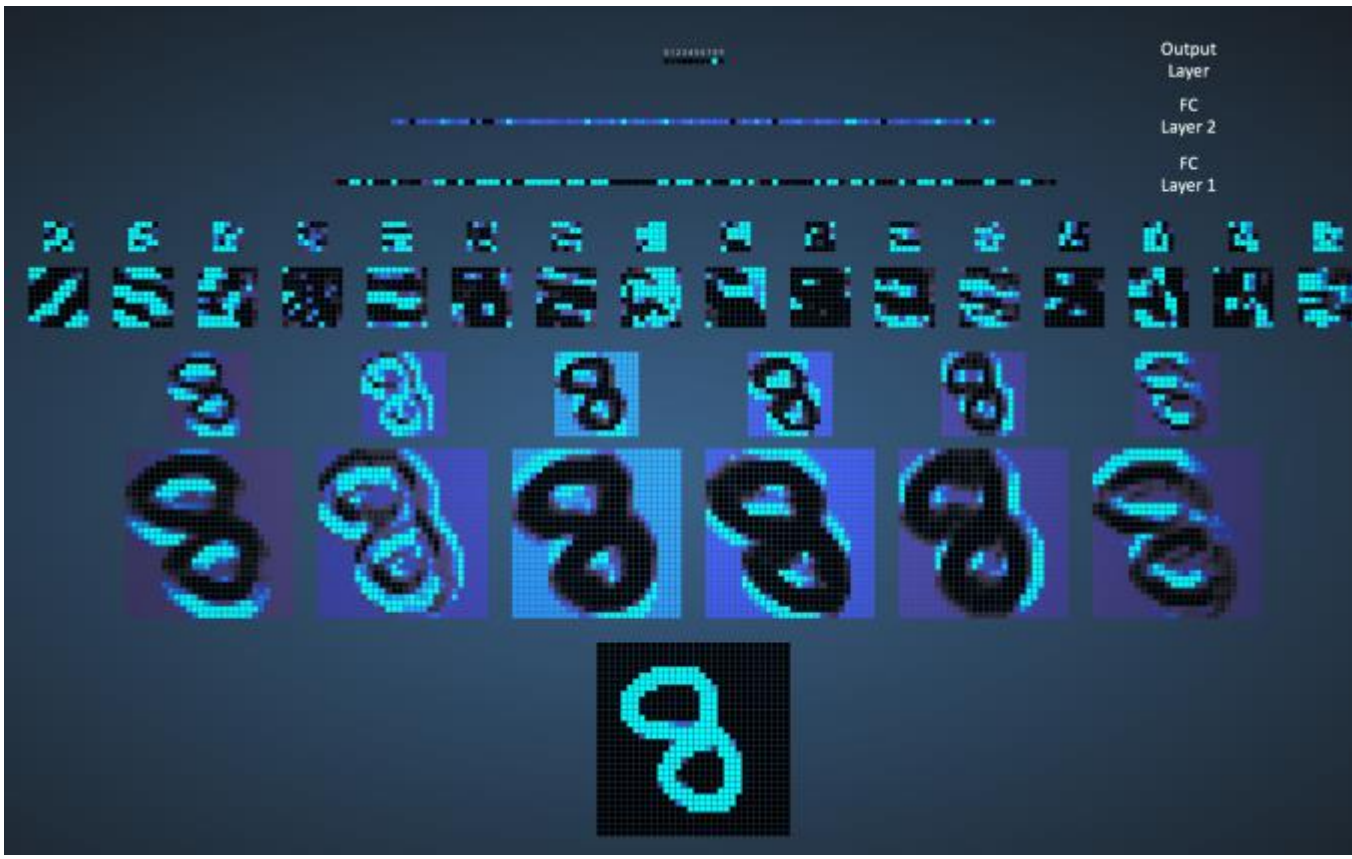


Figure 41: Visualizing a ConvNet on Handwritten Digits

The input image contains 1024 pixels (32 x 32 image) and the first Convolution layer (Convolution Layer 1) is formed by convolution of six unique 5 × 5 (stride 1) filters with the input image. As seen, using six different filters produces a feature map of depth six.

Convolutional Layer 1 is followed by Pooling Layer 1 that does 2 × 2 max pooling (with stride 2) separately over the six feature maps in Convolution Layer 1. You can move your mouse pointer over any pixel in the Pooling Layer and observe the 2 x 2 grid it forms in the previous Convolution Layer (demonstrated in **Figure 42**). You'll notice that the pixel having the maximum value (the brightest one) in the 2 x 2 grid makes it to the Pooling layer.

Figure 42: Visualizing the Pooling Operation

Pooling Layer 1 is followed by sixteen 5 × 5 (stride 1) convolutional filters that perform the convolution operation. This is followed by Pooling Layer 2 that does 2 × 2 max pooling (with stride 2). These two layers use the same concepts as described above.

We then have three fully-connected (FC) layers. There are:

- 120 neurons in the first FC layer
- 100 neurons in the second FC layer
- 10 neurons in the third FC layer corresponding to the 10 digits – also called the Output layer

Notice how in **Figure 43**, each of the 10 nodes in the output layer are connected to all 100 nodes in the 2nd Fully Connected layer (hence the name fully Connected).

Also, note how the only bright node in the Output Layer corresponds to '8' – this means that the network correctly classifies our handwritten digit (brighter node denotes that the output from it is higher, i.e. 8 has the highest probability among all other digits).

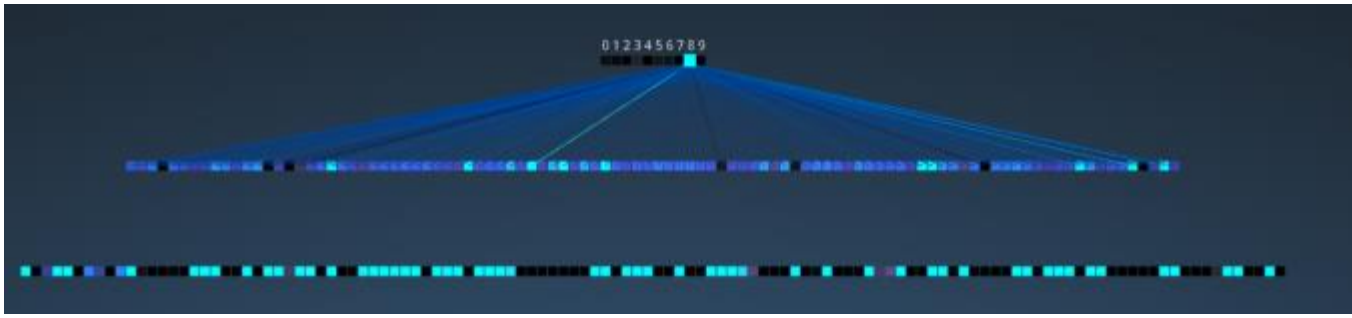


Figure 43: Visualizing the Filly Connected Layers

3.3.3 Other ConvNet Architectures

Convolutional Neural Networks have been around since early 1990s. We discussed the LeNet above which was one of the very first convolutional neural networks. Some other influential architectures are listed below.

- **LeNet (1990s):** Already covered in this article.
- **1990s to 2012:** In the years from late 1990s to early 2010s convolutional neural network were in incubation. As more and more data and computing power became available, tasks that convolutional neural networks could tackle became more and more interesting.
- **AlexNet (2012) –** In 2012, Alex Krizhevsky (and others) released AlexNet which was a deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was a significant breakthrough with respect to the previous approaches and the current widespread application of CNNs can be attributed to this work.
- **ZF Net (2013) –** The ILSVRC 2013 winner was a Convolutional Network from Matthew Zeiler and Rob Fergus. It became known as the ZFNet (short for Zeiler & Fergus Net). It was an improvement on AlexNet by tweaking the architecture hyper parameters.
- **GoogLeNet (2014) –** The ILSVRC 2014 winner was a Convolutional Network from Szegedy et al.from Google. Its main contribution was the development of an *Inception Module* that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M).
- **VGGNet (2014) –** The runner-up in ILSVRC 2014 was the network that became known as the VGGNet. Its main contribution was in showing that the depth of the network (number of layers) is a critical component for good performance.
- **ResNets (2015) –** Residual Network developed by Kaiming He (and others) was the winner of ILSVRC 2015. ResNets are currently by far state of the art

Convolutional Neural Network models and are the default choice for using ConvNets in practice (as of May 2016).

- **DenseNet (August 2016) –** Recently published by Gao Huang (and others), the Densely Connected Convolutional Network has each layer directly connected to every other layer in a feed-forward fashion. The DenseNet has been shown to obtain significant improvements over previous state-of-the-art architectures on five highly competitive object recognition benchmark tasks.

3.3.4 Object recognition applications:

Face detection

Popular applications include face detection and people counting.

People counting

Object detection can be also used for people counting, it is used for analyzing store performance or crowd statistics during festivals. These tend to be more difficult as people move out of the frame quickly (also because people are non-rigid objects).

Vehicle detection

Similarly when the object is a vehicle such as a bicycle or car, object detection with tracking can prove effective in estimating the speed of the object. The type of ship entering a port can be determined by object detection (depending on shape, size etc.) This system for detecting ships are currently in development in some European countries.

Manufacturing Industry

Object detection is also used in industrial processes to identify products. Say you want your machine to only detect circular objects. Hough circle detection transform can be used for detection.

Online images

Apart from these object detection can be used for classifying images found online. Obscene images are usually filtered out using object detection.

Security

In the future we might be able to use object detection to identify anomalies in a scene such as bombs or explosives (by making use of a quadcopter).

CHAPTER 4:
The Web Interface

4.1 HTTP Protocol

HTTP stands for Hypertext Transfer Protocol. It's a stateless, application-layer protocol for communicating between distributed systems, and is the foundation of the modern web. As a web developer, we all must have a strong understanding of this protocol.

4.1.1 HTTP Basics

HTTP allows for communication between a variety of hosts and clients, and supports a mixture of network configurations.

This makes HTTP a stateless protocol. The communication usually takes place over TCP/IP (Transmission Control Protocol/Internet Protocol), but any reliable transport can be used. The default port for TCP/IP is 80, but other ports can also be used.

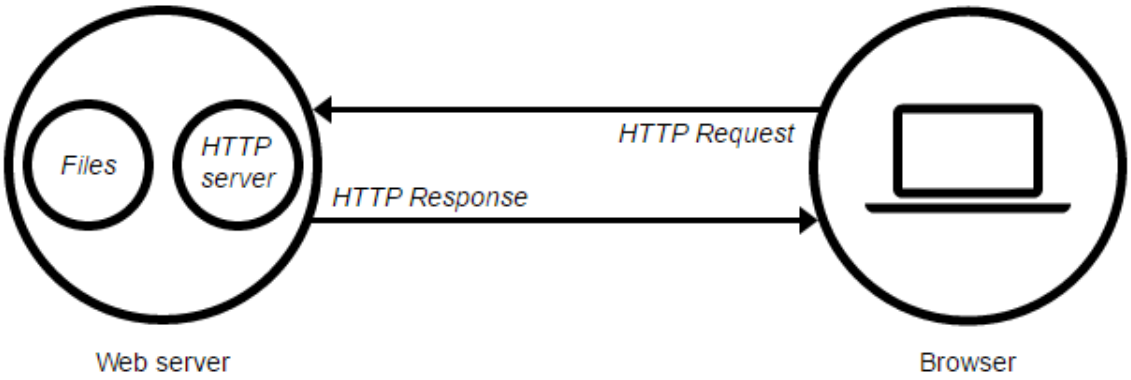


Figure 44: HTTP Protocol

Custom headers can also be created and sent by the client.

Communication between a host and a client occurs, via a request/response pair. The client initiates an HTTP request message, which is serviced through a HTTP response message in return. We will look at this fundamental message-pair in the next section.

The current version of the protocol is HTTP/1.1, which adds a few extra features to the previous 1.0 version. The most important of these, in my opinion, includes persistent connections, chunked transfer-coding and fine-grained caching headers. We'll briefly touch upon these features in this article; in-depth coverage will be provided in part two.

4.1.2 URLs

At the heart of web communications is the request message, which are sent via Uniform Resource Locators (URLs). URLs have a simple structure that consists of the following components:

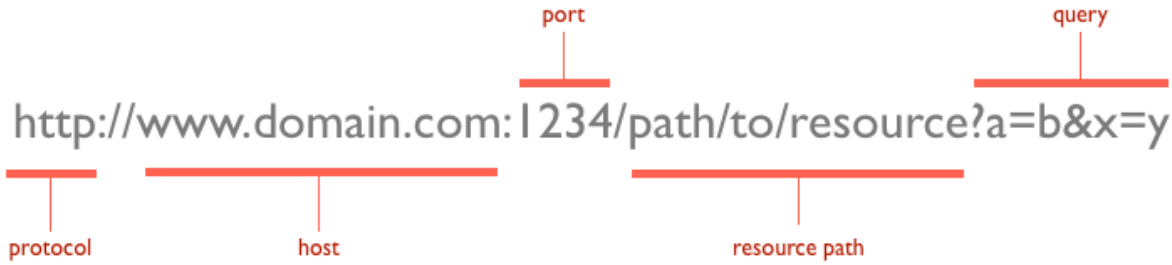


Figure 45: URLS Structure

The protocol is typically http, but it can also be https for secure communications. The default port is 80, but one can be set explicitly, as illustrated in the above image. The resource path is the local path to the resource on the server.

4.1.3 Verbs

URLs reveal the identity of the particular host with which we want to communicate, but the action that should be performed on the host is specified via HTTP verbs. Of course, there are several actions that a client would like the host to perform. HTTP has formalized on a few that capture the essentials that are universally applicable for all kinds of applications.

These request verbs are:

GET: fetch an existing resource. The URL contains all the necessary information the server needs to locate and return the resource.

POST: create a new resource. POST requests usually carry a payload that specifies the data for the new resource.

PUT: update an existing resource. The payload may contain the updated data for the resource.

DELETE: delete an existing resource.

The above four verbs are the most popular, and most tools and frameworks explicitly expose these request verbs. PUT and DELETE are sometimes considered specialized versions of the POST verb, and they may be packaged as POST requests with the payload containing the exact action: create, update or delete.

4.2 Web Servers

A web server is a computer system that processes requests via HTTP, The primary function of a web server is to store, process and deliver web pages to clients.

4.2.1 Web server's definition

A web server can refer to hardware or software, or both of them working together.

On the hardware side, a web server is a computer that stores a website's component files (e.g. HTML documents, images, CSS stylesheets, and JavaScript files, not covered in this report) and delivers them to the end-user's device. It is connected to the Internet and can be accessed through a domain name like mozilla.org.

On the software side, a web server includes several parts that control how web users access hosted files, at minimum an HTTP server. An HTTP server is a piece of software that understands URLs and HTTP.

At the most basic level, whenever a browser needs a file hosted on a web server, the browser requests the file via HTTP. When the request reaches the correct web server (hardware), the HTTP server (software) sends the requested document back, also through HTTP.

Figure 46: Web Servers

4.2.2 Types of web servers

To publish a website, you need either a static or a dynamic web server.

A static web server, or stack, consists of a computer (hardware) with an HTTP server (software). We call it "static" because the server sends its hosted files "as-is" to your browser.

A dynamic web server consists of a static web server plus extra software, most commonly an application server and a database. We call it "dynamic" because the

application server updates the hosted files before sending them to your browser via the HTTP server.

For example, to produce the final webpages you see in the browser, the application server might fill an HTML template with contents from a database. Sites like Wikipedia have many thousands of webpages, but they aren't real HTML documents, only a few HTML templates and a giant database. This setup makes it easier and quicker to maintain and deliver the content.

4.2.3 Famous web servers

There are 4 primary web servers:

Apache (provided by Apache)

IIS (provided by Microsoft and short for Internet Information Server)

Nginx (provided by NGINX, Inc. and pronounced like “Engine X”)

And GWS (provided by Google and short for Google Web Server)

Currently, Apache is the most popular with IIS gaining in popularity soon becoming the most popular web server. Nginx is an extremely popular alternative as it is very fast and very lightweight, while GWS is the least used with a small percentage of use.

4.3 Databases

Organized collection of data is called database. It is really important to organize data and different database model uses unique processes to store large data sets using processes requiring information. For example, the availability of a table in a hotel can be measured by the vacancies.

4.3.1 Database management systems

Database management systems (DBMSs) are computer software applications that interact with the user define programs, applications, and metadata to capture and analyze data. A general management purpose DBMS is designed to allow structure and design to any data by using some administration protocols of databases.

Some of the well-known DBMSs include MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SAP and DB2. So, a database is a place to store and retrieve organized data. In practical usage, web hosting, databases store your personal blog data, the date they were live, the opinions people posted with them, and your administrative information. It's all stored in the database classified according to the database model that they support.

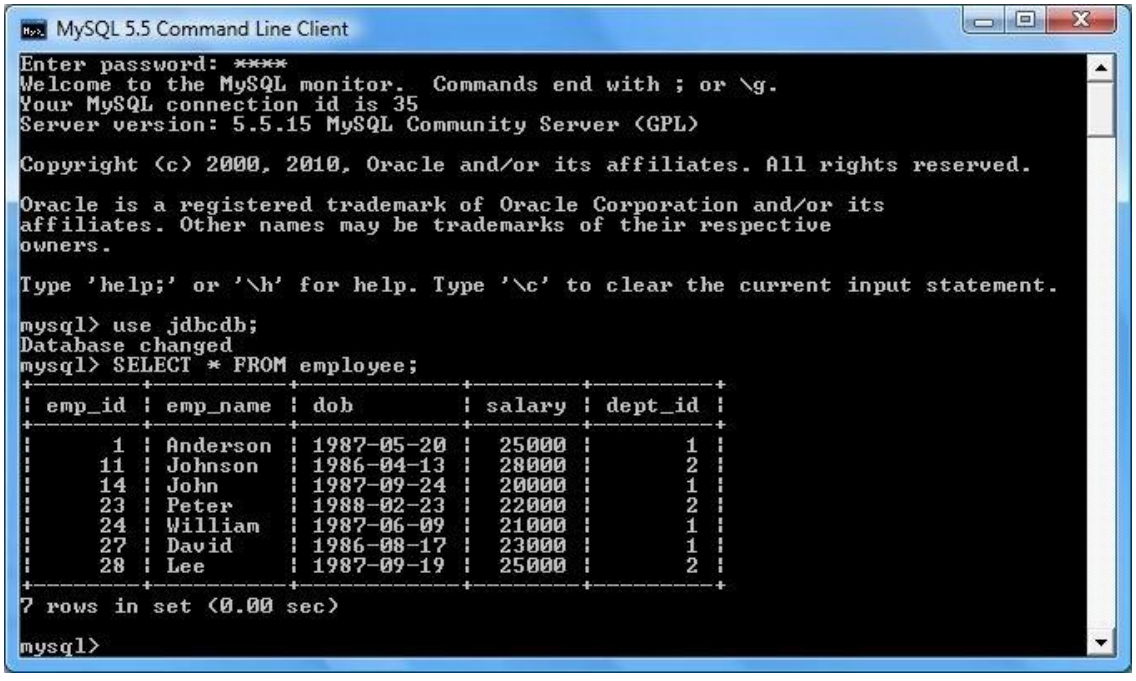


Figure 47: MYSQL Over CMD

Using program, According to its characteristics SQLite is a popular choice as an embedded database for local/client storage in application software such as different web browsers. It is perhaps the most widely deployed database server engine used today not only by all leading browsers, operating systems but also by embedded systems, among others.

4.4 LAMP Stack

LAMP is an archetypal model of web service stacks, named as an acronym of the names of its original four open-source components: the Linux operating system, the Apache HTTP Server, the MySQL relational database management system (RDBMS), and the PHP programming language. The LAMP components are largely interchangeable and not limited to the original selection. As a solution stack, LAMP is suitable for building dynamic web sites and web applications.

4.4.1 Exploding the Acronym

Simply exploding the acronym on a letter by letter basis gives us the following elements:

Linux

Apache Web server

MySQL database

Perl, Python, or PHP

Individually, each of these items is a powerful component in its own right. The key to the idea behind LAMP, a term originally coined by Michael Kunze in the German magazine c't in 1998, is the use of these items together. Although not actually designed to work together, these open source software alternatives are readily and freely available. This has led to them often being used together. In the past few years, their compatibility and use together has grown and been extended. Certain extensions have even been created specifically to improve the cooperation between different components.

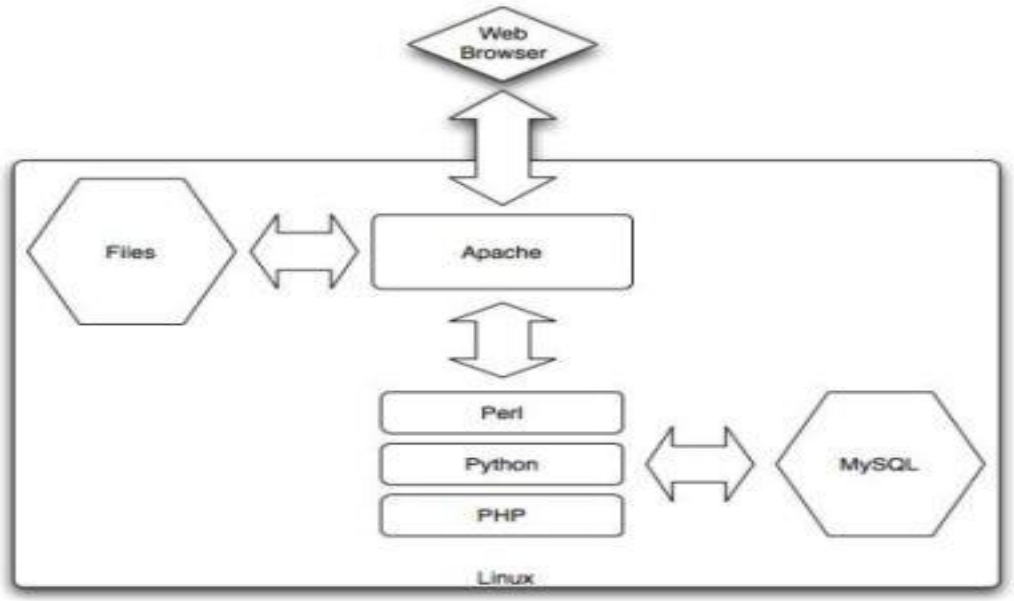


Figure 48: The lamp Stack

Each of the components in the LAMP stack is an example of Free or Open Source Software (FOSS). The benefit of the FOSS approach is three-fold. First, the nature of FOSS software means applications are available for free download, making them readily available to a wide range of people without payment. That makes the software incredibly attractive to a wide range of users who would otherwise have to pay for "professional" commercial tools, which is often an expensive step in producing a Web site.

Second, FOSS licenses are open and thus have few restrictions on their use and deployment of applications based on the FOSS technology. It is possible to develop and deploy LAMP-based projects without paying any license fees for distributing the software, and this, again, makes it popular for both hobbyists and professionals alike.

Third, and a major reason for the growth and use of FOSS technology (including LAMP), is that because users have access to the source it is much easier to fix faults and improve the applications. In combination with the open license, this simplifies the development process for many enterprises and gives them flexibility that simply isn't available within the confines of a proprietary or commercial-based product.

4.4.2 Using the LAMP Stack

Choosing to use LAMP in your business is not about cost — although many enterprises will be attracted to the low-cost required for both development and deployment. Instead, choosing LAMP for your organization is about the benefits it provides, as summarized below.

Flexibility: There are no limits to what you can do with the LAMP stack, either technically or because of licensing restrictions. This allows you the flexibility to build and deploy applications in a method that suits you, not the supplier of the technology you are using.

Customization: Because LAMP components are open source, they have built up a huge array of additional components and modules that provide additional functionality. The open source approach enables you to do the same, customizing components and functionality to suit your needs.

Ease of Development: You can write powerful applications using LAMP technology in relatively few lines of code. Often the code is straightforward enough that even nonprogrammers can modify or extend the application.

Ease of Deployment: With neither licensing issues nor the need to compile applications, deployment is often as easy as copying an application to a new host. Most hosting services provide LAMP-based environments as standard, or they can be deployed using a Linux distribution, such as Fedora or Debian.

Security: With many eyes developing the software and years of use by a wide range of users and community groups, LAMP technology is secure and stable. Problems are normally fixed very quickly, and without the need for a costly support contract.

Community and Support: A wide and experienced group of people is willing to provide help and support during the development and deployment of LAMP-based applications.

Many successful business have already leveraged the use of LAMP technology. Many heavily trafficked Web sites use LAMP, or components of it, to support their applications.

CHAPTER 5

IoT Robot Interface

5.1 Stack Infrastructure Components

A robot that is controlled manually by a human needs some kind of interface in order to control it. The type of robot, its functions, technologies used and purpose all decide what kind of interface to use. For a robot that will be used form huge distances with continuous stream of data, connecting the robot to an internet gateway will provide access to the robot from anywhere in the world giving it the power to be used in really remote locations as long as it's connected to internet. The technology used to connect to internet (e.g. wifi, satellite, etc.) is not a concern in the point of view of user interface, but the characteristics of the web application are the main concern here.

5.1.1 Goals to achieve

In order to define the web app characteristics, actions needed to be done by the interface have to be specified first. There are multiple actions need to be done with the interface that can be listed as:

Receiving the robot state which can be divided as:

Getting the current moving state

Getting the camera stream

Getting the current object recognition results for a specific frame

Sending commands to the robot, which is essentially asking to receive one of the robot states at a given time, divided as:

Sending move instructions to robot

Request robot camera stream

Request recognition of objects in a specific frame

5.1.2 Characteristics of the web interface

We can now define the web app characteristics as:

Single user: The robot can only be controlled by 1 person at a time.

Real-time System: The app only deals with real-time data and states and doesn't store any previous or old data or states

Direct access to hardware drivers: The interface has to request binary stream data from a camera and give movement instructions to motors

5.1.3 Infrastructure components Decisions

Infrastructure for web applications is the same as the web stack used including the hosting machine containing the stack. Follows are each of the components and the candidates that can be used and which candidate is the best to be used.

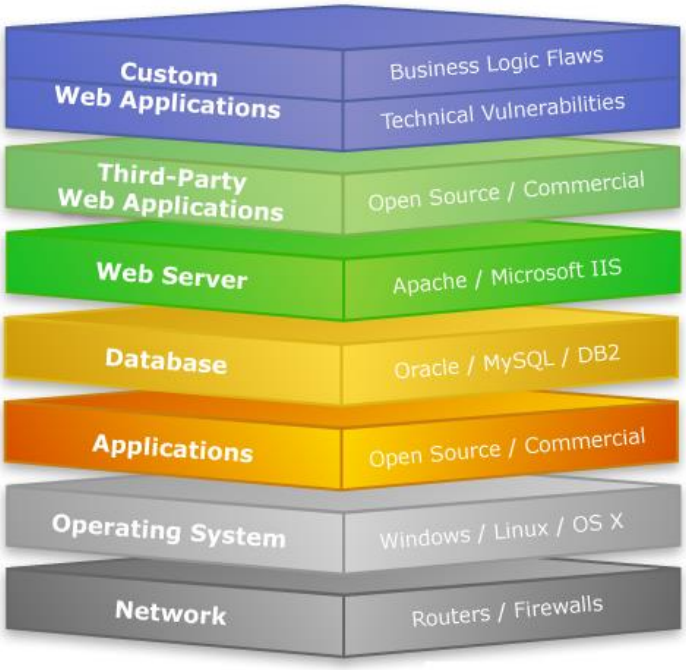


Figure 49: Web Stack Components

5.1.3.1 Hosting machine

The hosting machine is the hardware that serves the web app and runs its stack. There are two options for the hosting machine of the app:

Use a hosting machine from a datacenter

Use the raspberry pi as a the hosting machine

The datacenter basically has more CPU power, more memory, more disk space and a huge bandwidth capacity. The raspberry pi in the other hand has limited resources in all these aspects and also the robot applications obligates it to connect to internet wirelessly which increases the chance of packet loss making the bandwidth smaller than what is intended to be.

Figure 50: Servers inside a Datacenter

Now we show a comparison between each selection in respect to the web app characteristics:

Being a single user system:

That means that the app won't use many resources from the machine. As only a very limited number of requests are issued at a time. Datacenters are designed to be able to handle hunders of thousands of requests which is a huge wasted resources for such a simple app, which makes the raspberry pi resources enough for this kind of application.

Being a Real-time System & having access to hardware drivers

Real-time systems need to have minimum delay as possible. Since the datacenters has huge bandwidths we can initially say that the datacenters would be more efficient, but then the datacenter will have to route requests again to the pi in order to access the hardware drivers making it a 2 rounds trip which adds an overhead of intermediate server communication to the initial cost of communicating with the pi directly. Which in overall add delay to the whole process, which is not wanted.

In general, even if a hosting machine within a datacenter is used, the needed for another server on raspberry pi is still needed to control the robot. And as the shortest route between 2 points is a straight line, having the raspberry pi as the hosting machine directly is or best option here.

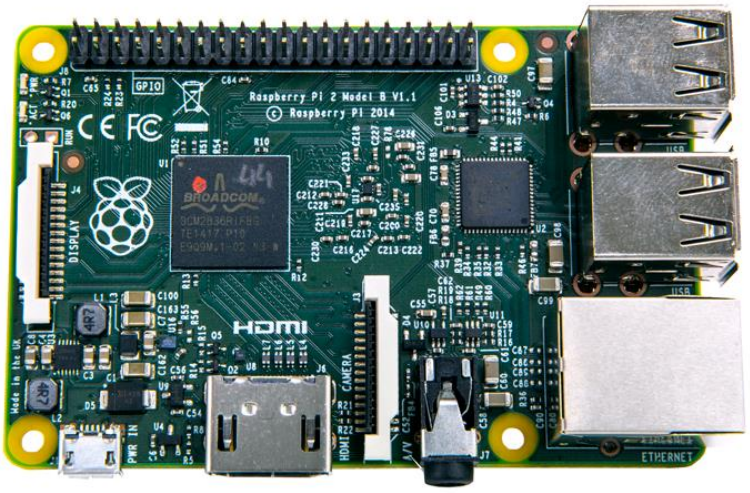


Figure 51: Raspberry Pi Board

5.1.3.2 Operating system

Choosing the raspberry pi as out hosting machine limits the options we have on choosing the operating system used to hold the stack to light-weight operation systems. As there is many solutions to use any preferred stack in any operating system, this choice is not really a problem here. The only characteristic that concerns us here is the accessibility of the hardware in real-time. The raspberry pi supports using Raspbian OS which is a linux-debian based operating system and has great support especially with pi-specific hardware like the raspberry pi camera. That's why Raspbian OS is used for this project.

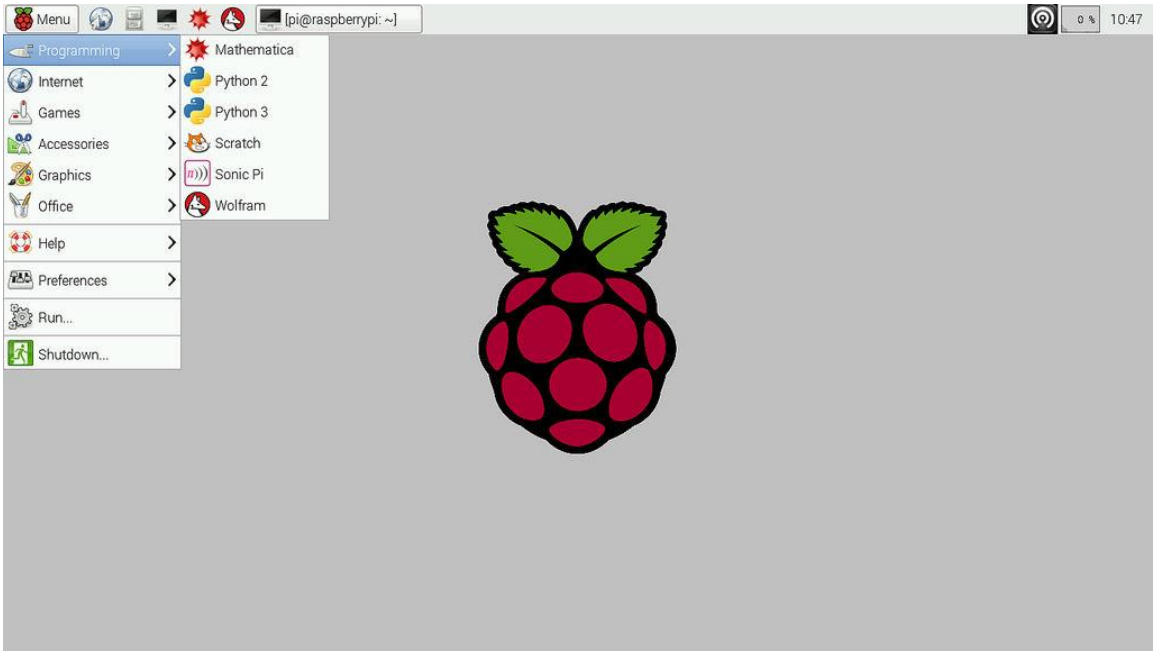


Figure 52: Raspbian OS GUI

5.1.3.3 Server-side scripting language

The server-side scripting language is the language used to write the server logic on the hosting machine, or the 'backend' of our web app. It doesn't include any scripting or markup languages used to build the user interface which is run on the user's browser, or as known as the 'frontend'.

There are many options here like PHP, Ruby, Python, NodeJS, etc. All of these would have no problem with handling a single user or be used in a real-time application.

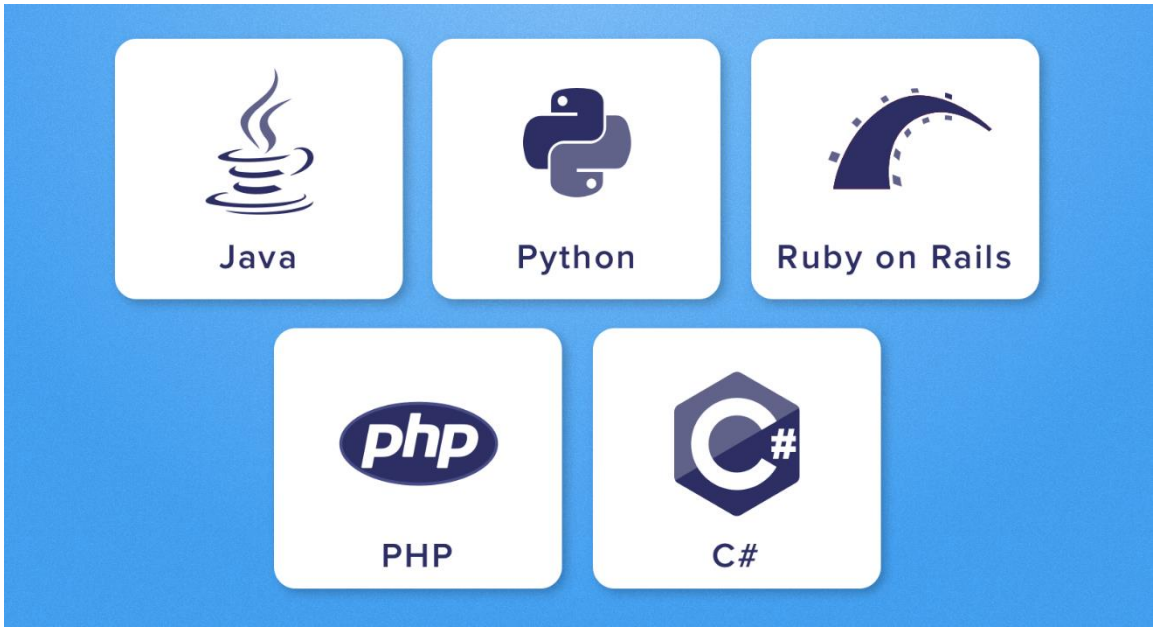


Figure 53: Server Side Scripting Languages

Raspberry pi offers great support for python in accessing its own hardware easily and efficiently. This makes python one of the best candidates that can combine between processing web requests and controlling system hardware like motors or camera without any need of external scripts which would add an overhead to every request.

Even if Python doesn't have native support for web applications like PHP for example and may not have the speed of other scripting languages like Ruby but having direct access to raspberry pi GPIO, I2C lines and serial camera makes Python the best scripting language to be used in such an application without much competition.

5.1.3.4 Web Server

There are different types of web servers available like Apache, IIS, Nginx and LiteSpeed. Mainly the web server listens to HTTP requests sent to a port and runs a script based on the URL specified in the HTTP request and returns the output of the script in an HTTP response. The importance of web servers can be abstracted in two main functions: HTTP handling and threading. Threading is so important in web apps as it allows the web server to handle many requests at the same time allowing many users to use the applications in parallel.

It's obvious that one of the main functions of web servers is not really needed because one of the main characteristics of the application is that it's only used by a single user at a time. That allows us to use a simpler web server that can save more resources than used by its competitors, especially on a limited resources device as raspberry pi. As python is used as the server-side scripting language, Python offers many solutions that can be used to handle light, small requests at a time. So the flask development server was used to easily and efficiently run our web app on the raspberry pi.

5.1.3.5 Database

One of the main characteristics of the application is that it's a real-time application that doesn't store any previous or old data or states. That means there is no need to add a database to our stack at all. This is more suitable to the limited space presented by the raspberry pi which is most used anyway for the recognition feature which is discussed in a different section.

5.2 Robot control interface

The control interface is the graphical interface that the user used to control the robot over the internet. Let's recall the main functions the interface has to be capable of doing:

Receiving the robot state:

Getting the current moving state

Getting the camera stream

Getting the current object recognition results for a specific frame

Sending commands to the robot:

Sending move instructions to robot

Request robot camera stream

Request recognition of objects in a specific frame

5.2.1 Interface components

Putting the required actions in mind, the interface can be divided into 4 parts:

The video stream: which show what's in front of the robot to be able to control it remotely

The robot controls: which are divided into:

Movement controls: which control the robot movement

Recognition button: which orders the robot to recognize what is in front of it

State display: which shows the current states of all the robot functions

Below is the control interface built, showing all of its components used to achieve the desired functions.

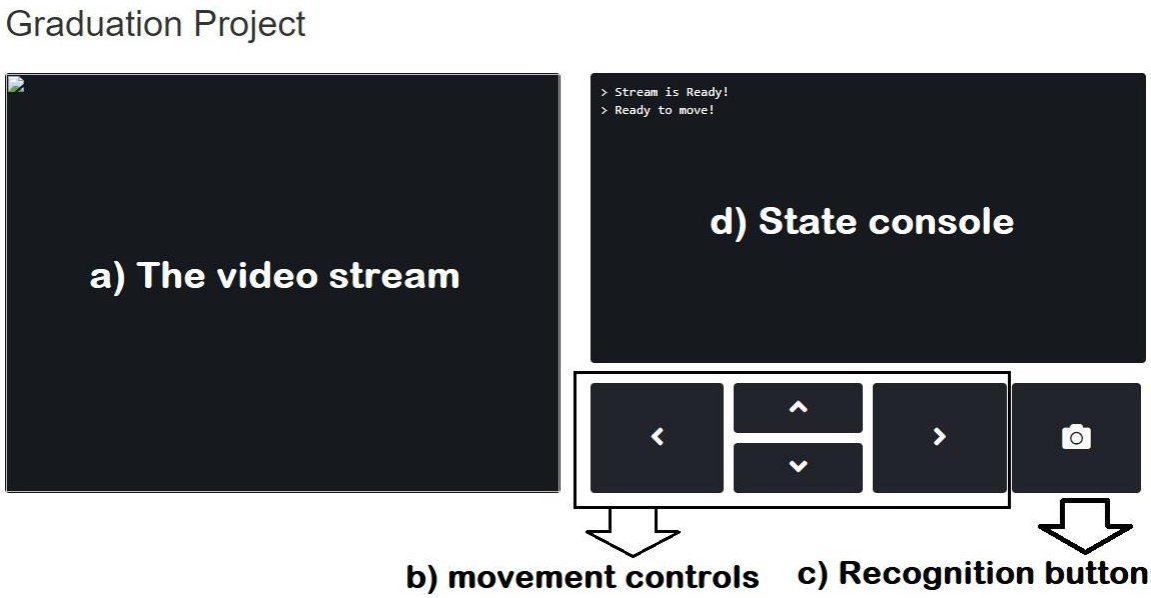


Figure 54: Control Interface Design

a) Video stream, b) movement controls,
c) recognition button, d) state console

5.2.2 Interface layers

The control interface is constructed as a web page and can be accessed through a browser. The interface can be divided into 3 main layers: The structure layer, the presentation layer and the behavior layers:

The structure or content layer

The Presentation layer

The Behavior layer

Figure 55: Webpage Layers

5.2.2.1 Structure layer

The structure or content layer of a web page is the underlying HTML code of that page. HTML consists of a series of short codes typed into a text file, these are the tags. A browser reads the file and translates the text into a visible form. The structure layer is just as a house's frame creates a strong foundation upon which the rest of the house is built, a solid foundation of HTML creates a platform upon which a website can be created.

The structure layer is where all the components of the interface are stored. HTML structure can consist of text and images, and can include hyperlinks to navigate through the interface. This is coded in standards-compliant HTML5 and can include text, images, and multimedia (video, audio, etc.).

Every aspect of a site's content should be represented in the structure layer, including other layers resources as well. This allows users who have JavaScript turned off or who can't view CSS access to the entire website, if not all of its functionality.

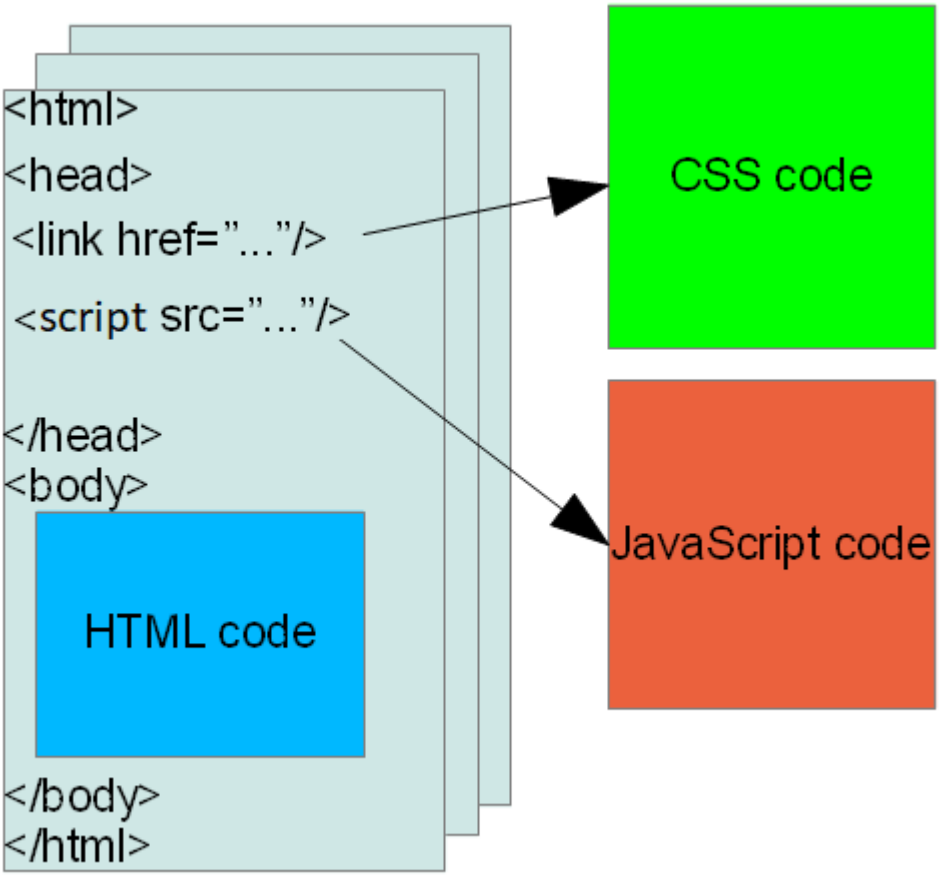


Figure 56: Example of HTML Document

Below is a screenshot of the control interface structure without using any styles at all

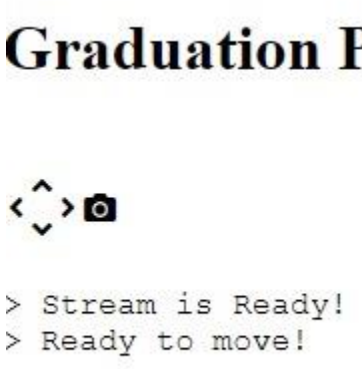


Figure 57: Control Interface without Styles

All the interface components are presented like the movement arrows and the capture button and state console. But there are no visible containers to present the components clearly. Also the stream section is not visible at all yet.

5.2.2.2 Presentation layer

This layer dictates how a structured HTML document will look to a user and is defined by CSS (Cascading Style Sheets). These files contain stylistic instructions for how the document should be displayed in a web browser. The style layer usually includes media queries that change a site's display based on screen size and device.

All visual styles for a website should reside in an external style sheet. Multiple style sheets can be used but are not needed for a 1 page interface.

Below is a screenshot of the control interface after applying styles to it

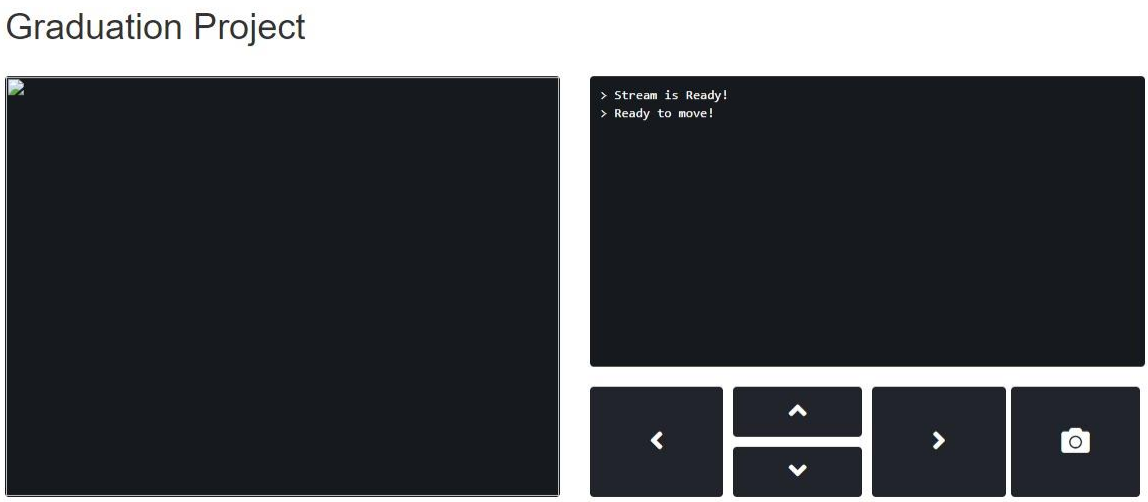


Figure 58: Control Interface with Styles on Desktop

Now all the components of the interface are clear and can be used more easily even for a person who uses this interface for the first time.

The interface looks great and very usable on a desktop machine with a large screen, but it's not the same for tablets and mobile phones; which are mostly used to control the robot. That's why we have to add 'responsiveness' to the control interface.

Responsiveness in the presentation layer means the ability to use the interface in any given device or screen size. That's why 'bootstrap' is used in the interface. Bootstrap is a front-end web framework that was created by Twitter for faster creation of device responsive web applications.

Below is a screenshot of the control interface on a smart phone after using bootstrap to make it responsive

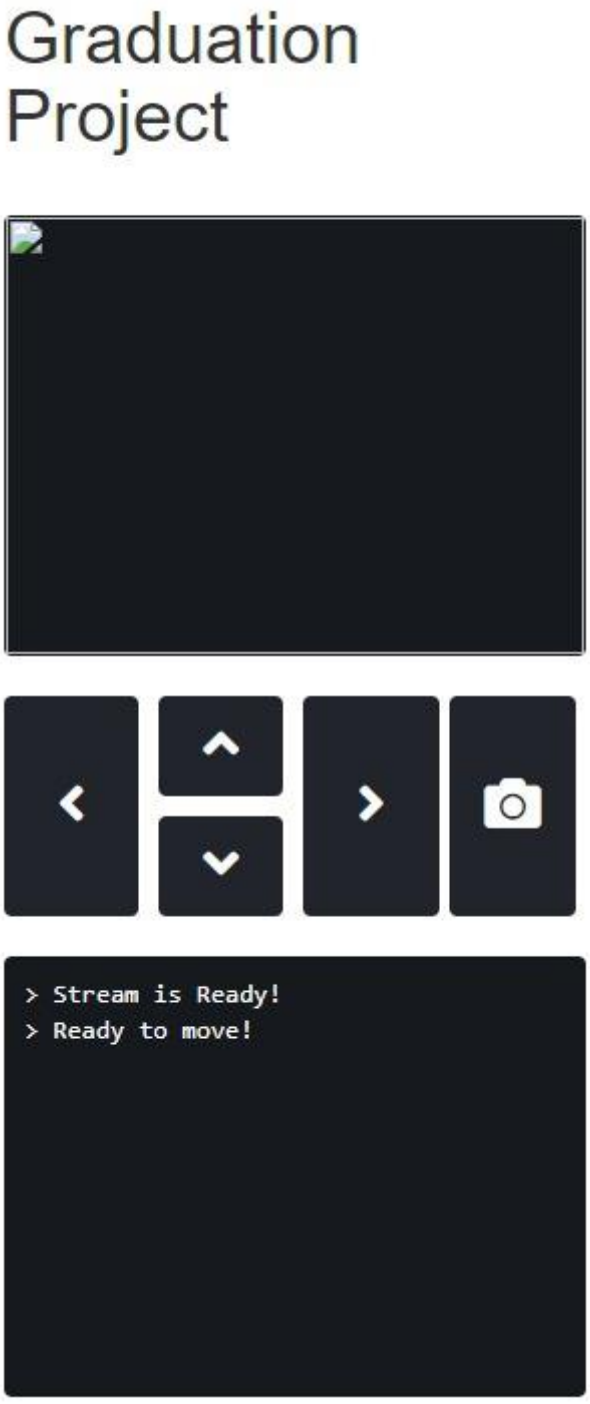


Figure 59: Control Interface on a Smart Phone

5.2.2.3 Behavior layer

The behavior layer makes a website interactive, allowing the page to respond to user actions or to change based on a set of conditions. JavaScript is the most commonly used language for the behavior layer.

When developers refer to the behavior layer, most of them mean the layer that is activated directly in the web browser. This layer is used to interact directly with the DOM (Document Object Model). Writing valid HTML in the content layer is important for DOM interactions in the behavior layer. An example of such an interaction is clicking on a direction arrow on the document, which is considered a DOM element; the behavior layer will interact with that action by sending a request to the robot to move at the same direction is the clicked direction.

The behavior layer has 3 responsibilities in the robot control interface:

Capture user interactions: it's used to capture mouse clicks and keyboard pressed keys and bind those events to a specific behavior (e.g. send a request to the robot to move)

Highlight user actions: It manipulate DOM elements to indicate that a certain event has happened like changing the style of a pressed button or write message on the console

Check robot availability: The robot, as well as the user's device, might experience connection losses. It would be inconvenient to reload the interface every time such a loss happens. That's why the behavior layer is responsible of checking the robot state and attempt to reconnect when the connection is lost at any given time.

It's clear that the behavior layer is responsible of all the communication between the frontend in the user's device and the robot itself. Using asynchronous requests to the server in order to not freeze all functions in the interface till one request is finished (aka. AJAX).

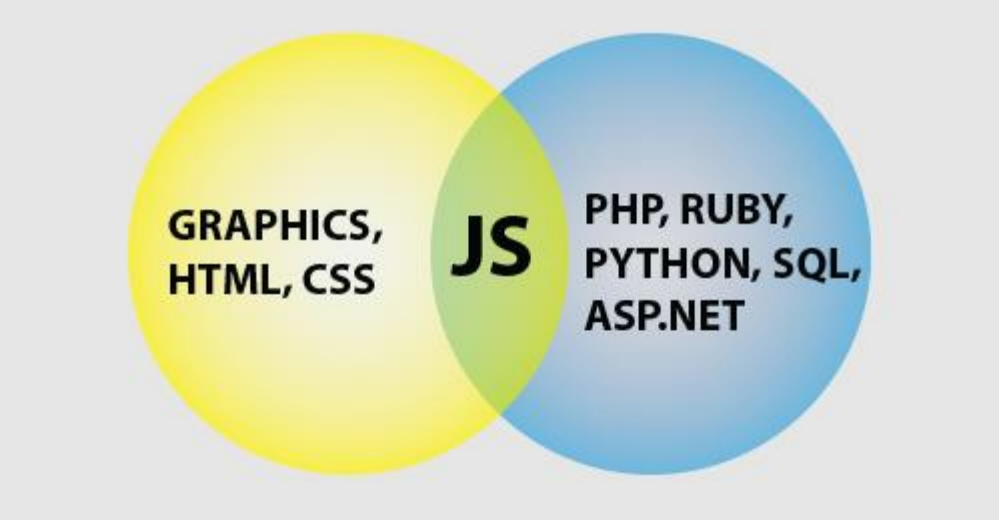


Figure 60: Fronted and Backend Interactions in WebApp

This layer is the layer that requests and displays all functions mentioned before and required for the interface and the robot to function correctly.

5.3 Server-side Robot Logic

The control interface of the robot is considered the frontend in web terminologies as it is what the user interacts with; but it doesn't perform the actual functions of the robot. The interface captures the user interactions the movement controls and sends a request to the robot to move accordingly but it doesn't move the robot itself. The same goes for video stream and object recognition; the interface only requests the function but doesn't perform it itself.

The server-side script written is the one responsible of receiving and processing HTTP requests and interacting with the hardware drivers and does the actual function required; which is considered the backend of the application.

5.3.1 Backend functionalities

The backend consists of a flask python app. That script is responsible of doing all the functionalities of the system; these functionalities are detailed as:

Serving the control interface upon request

Capture camera frames and send it back to requests as stream

Send movement instructions to the underlying microcontroller via a common communication protocol

Recognize objects in a frame and send the results back upon request

It's clear that the app only do a function only if it received a request to do it, that's why it's basically called a 'server'. All of these functions required direct access to hardware drivers; which Python presents especially on raspberry pi devices.

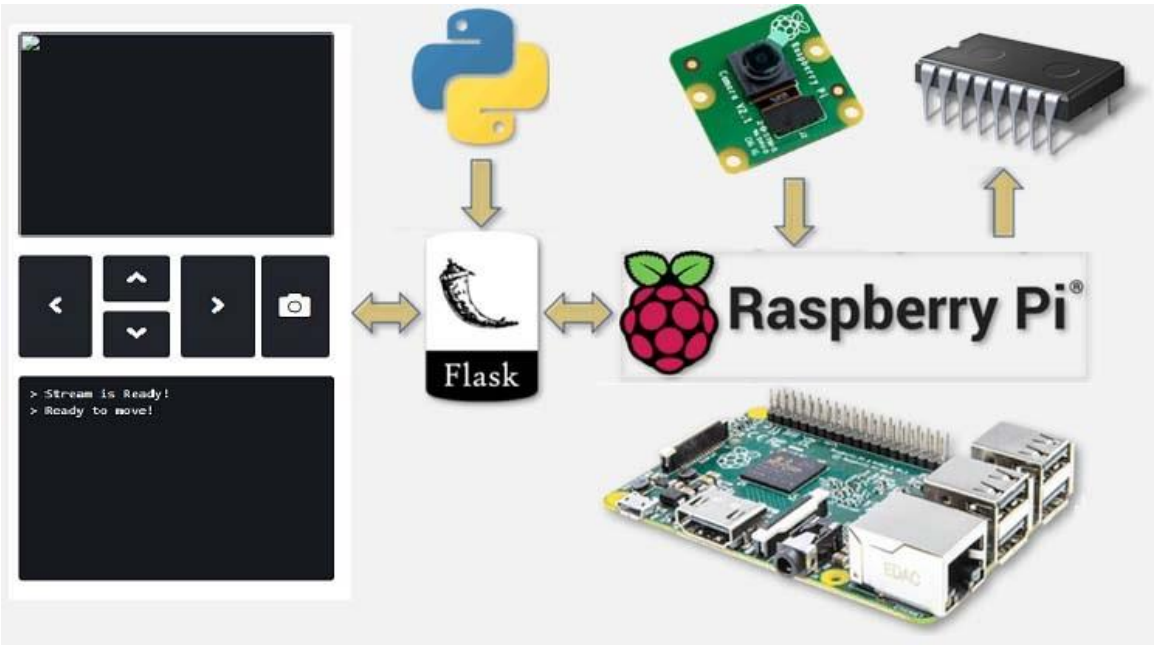


Figure 61: Raspberry Pi Interactions

5.3.1 Backend structure

The structure of the backend can be divided into 2 parts.

Files structures: Which represents the separation of concerns in the application

Routes structure: Which is represented by the HTTP URLs used to access different robot functionalities

The files structure is listed below; but it's not our concern for now as it might change without affecting the way the robot behaves at all.

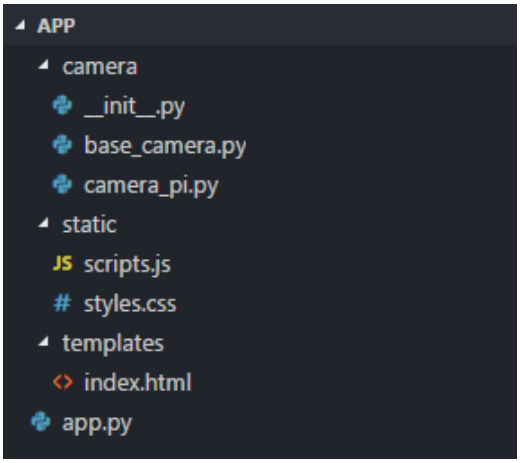


Figure 62: Backend Files Structure

Routes structure is the logical structure of paths used by the control interface to access specific function. Essentially each functionality the app does has its own HTTP route that browsers can send requests to and a for each route there is a function that process the HTTP request and serves the desired functionality back in a server-client way of communication.

5.3.1.2 How routes work

Each route constructs a URL. The server accepts specific HTTP verbs for each URL providing a standardized API (Application programming interface) to make communications clear and easy.

For example: If the raspberry pi has IP address of 192.168.0.1 and the web server listens to port number 5000 for HTTP requests, and the server serves the camera stream through the route `/video_feed` if it gets a GET HTTP request; then a user, or in our case the control interface, can access video stream by sending a GET request to:

http://192.168.0.1:5000/video_feed

Requests can be sent asynchronously to the backend using the control interface. Routes that accept GET requests like the root route and the video stream route can be accessed directly through a browser by visiting their URLs as described in the previous example.

5.3.1.3 Routes mapping

Follows are the routes structure for the backend app, showing the HTTP verb used to access them and the required query data to be sent in order for the route to response successfully. How each route does its function and how the response of each route is calculated is described with more detail in later sections.

`/` The root route
The root route is the route specified by only the domain name or IP address of the server with the port number only.

HTTP verb: GET

Required query data: None

Response: returns the control interface to the browser

`/video_feed` The stream route

HTTP verb: GET

Required query data: None

Response: returns multipart/x-mixed-replace HTTP response with an MJPEG images stream creating the video stream

`/move` The movement instructions gateway

HTTP verb: POST

Required query data: data specifying the required movements to be done

‘left’ (Boolean)

‘up’ (Boolean)

‘right’ (boolean)

‘down’ (boolean)

Response: JSON formatted response containing the current robot moving state

`/recognize` The stream route

HTTP verb: POST

Required query data: None

Response: JSON formatted response containing the results of the recognition operation

5.3.2 Functions implementation

This is the deepest point of the web application. The next layer is a hardware layer and a microcontroller layer. Follows are the detailed way of how the backend communicates with hardware to make a certain function happen.

5.3.2.1 Serving the control interface

The first thing the application does is to serve the robots control interface to the user in order to interact with the rest of the robot. This is done simply by sending a response containing the HTML document of the control interface.

Once the browser receives the HTML document, it will render it requesting any resources specified in the HTML document; which are basically:

CSS style sheets to handle the document presentation

JS scripts to handle events and interactions with the interface

The video streaming specified as a link of an image that should be fetched

The CSS and JS files are static files that the server serves to the user the same way it serves the HTML document of the control interface.

5.3.2.2 Video Streaming

Streaming was mentioned for awhile now without defining streaming in a technical way. Streaming is a technique in which the server provides the response to a request in chunks. Streams are useful in these cases:

Very large responses: Having to assemble a response in memory only to return it to the client can be inefficient for very large responses. An alternative would be to write the response to disk and then return the file but that adds I/O to the mix. Providing the response in small portions is a much better solution, assuming the data can be generated in chunks.

Real time data: For some applications a request may need to return data that comes from a real time source. An example of this is a real time video or audio feed. A lot of security cameras use this technique to stream video to web browsers.

An interesting use of streaming is to have each chunk replace the previous one in the page, as this enables streams to animate in the browser. With this technique you can have each chunk in the stream be an image, and that gives a video feed that runs in the browser.

The secret is to use a multipart HTTP response, which consist of a header that includes one of the multipart content types, followed by the parts, separated by a boundary marker and each having its own part specific content type.

There are several multipart content types for different needs. For the purpose of having a stream where each part replaces the previous part the `multipart/x-mixed-replace` content type must be used. This is the structure of a multipart video stream:

HTTP/1.1 200 OK

Content-Type: multipart/x-mixed-replace; boundary=frame

--frame

Content-Type: image/jpeg

<jpeg data here>

--frame

Content-Type: image/jpeg

<jpeg data here>

As seen above, the structure is pretty simple. The main Content-Type header is set to multipart/x-mixed-replace and a boundary string is defined. Then each part is included, prefixed by two dashes and the part boundary string in their own line. The parts have their own Content-Type header, and each part can optionally include a Content-Length header with the length in bytes of the part payload, but at least for images browsers are able to deal with the stream without the length.

There are many ways to stream video to browsers, and each method has its benefits and disadvantages. The method that works well with the streaming feature of Flask

and the application nature that needs to extract frames out of the stream to recognize objects in it is to stream a sequence of independent JPEG pictures. This is called Motion JPEG, and is used by many IP security cameras. This method has low latency, but quality is not the best, since JPEG compression is not very efficient for motion video.

Once the server receives a request for the video feed, it starts to create a multipart/x-mixed-replace HTTP response appending a camera frame as JPEG image to it followed by the boundary line and keeps updating and sending that response till the stream page is closed.

5.4 Hardware Interfacing

The robot has four layers of operation and three hardware layers for convenient connections and wiring.
Doing this gave us modularity in the software and the hardware allowing better

Figure 63: Front view of Top Layer

manipulation and control of the robot.

Figure 64: Top view of Mid layer

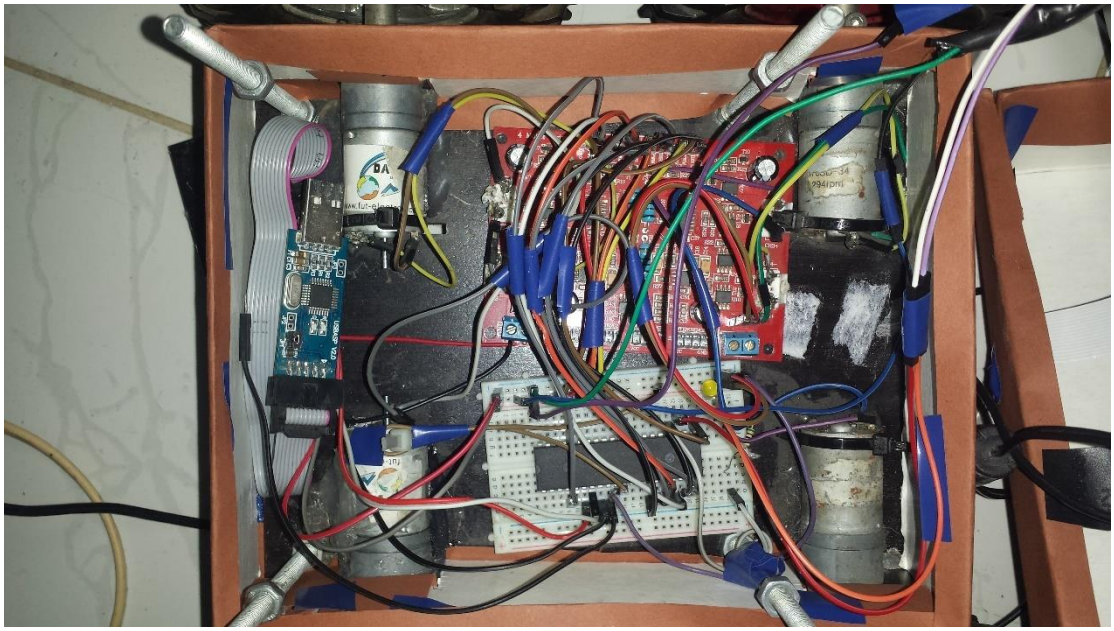


Figure 65: Top view of Base layer

5.4.2 Atmega32 with Motor Driver

- Four PWM pins (490HZ):

To control the speed of the four motors

- 4 output GPIO pins

To control the direction of the four motors

5.4.3 Atmega32 with Raspberry Pi

They are communicating to each other through TWI (Two wire interface) protocol which is known also as I2C, The raspberry pi is the master and the Atmega32 chip is the slave with an address of (33).

The Raspberry Pi is the master controller in our system so via this connection it can send commands to the Atmega32 chip to control the motors.

Recommendations and Future Work

Recommendations

First: We recommend using the google.ai TensorFlow library as a ready to use model that's already trained to huge data and characterized by accepted or better accuracy for object recognition

Second: Using the Raspberry pi was good enough for our project but it was not that efficient, In case it's possible to get another SoC with a higher processing power and speed, This would give great advantage for control and avoid image processing problems and live streaming lag or delay.

Third: The Atmega32 Microcontroller was more than enough for controlling the robot, It could successfully achieve it's function well with no problems so there is no need to use another controller, especially if it's more expensive.

Last: It's recommended to use a high resolution camera, This would not only give a much more fun and better experience for the live streaming, but also improve the functionality of the robot and make it easier to recognize objects using any library.

Future work

- Turning the robot autonomous:

It's possible to add few sensors and improve controlling mechanism of the robot and maybe even using error handling techniques like PID control to make our robot move by itself anywhere, Opening the field to much more useful applications.

- Human Assistant:

Adding a robot arm to our robot would give it unlimited number of applications.

- Area Scanning:

It's desired to upgrade our robot software and hardware to function better allowing it to be capable of scanning a room or a place and hence not only streaming that online but also giving a graphical output like a blueprint to that room or area.

Appendix

HTML Document (Interface Structure Layer)

```
<!DOCTYPE html>
<html lang="ar" dir="ltr">

<head>
  <title>
    Graduation Project
  </title>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <link rel="shortcut icon" type="imagex-icon" href="{{
url_for("static", filename="favicon.ico") }}" />
  <link rel="stylesheet" href="{{ url_for("static",
filename="libs/normalize.min.css") }}" />
  <link rel="stylesheet" href="{{ url_for("static",
filename="libs/bootstrap.min.css") }}" />

  <link rel="stylesheet" href="{{ url_for("static",
filename="styles.css") }}" />
</head>

<body class="">

  <div class="container">
    <div class="row">
      <div class="col-xs-12 section">
        <h1>Graduation Project</h1>
      </div>
    </div>
    <div class="row">
      <div class="col-sm-12 col-md-6 stream-container section">
        <img src="{{ url_for('video_feed') }}" data-src="{{
url_for('video_feed') }}">
      </div>
      <div class="col-sm-12 col-md-6">
        <div class="row">
          <div class="col-xs-12 section table-wrapper">
            <table class="table arrows">
              <tr>
                <td class="sides left-side" rowspan="2">
                  <span class="btn" id="left-arrow" data-arrow="left">
```

```
                    left
                  </span>
                </td>
                <td>
                  <span class="btn" id="up-arrow" data-arrow="up">
                    up
                  </span>
                </td>
                <td class="sides right-side" rowspan="2">
                  <span class="btn" id="right-arrow" data-
arrow="right">
                    right
                  </span>
                </td>
                <td class="sides" rowspan="2">
                  <span class="btn" id="recognize" data-arrow="enter">
                    detect
                  </span>
                </td>
              </tr>
              <tr>
                <td>
                  <span class="btn" id="down-arrow" data-arrow="down">
                    down
                  </span>
                </td>
              </tr>
            </table>
          </div>
          <div class="col-xs-12 section console-wrapper">
            <div class="console">
              <pre></pre>
            </div>
          </div>
        </div>
      </div>
    </div>
    <div class="row">
      <div class="col-xs-12" id="error_container">

      </div>
    </div>
  </div>

  <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"
></script>
```

```
  <script type="text/javascript" src="{{ url_for("static",
filename="scripts.js") }}"></script>
</body>

</html>
```

Cascade Style Sheet (Interface Presentation Layer)

```
body {
    min-height: 100vh;
}
img {
    max-width: 100%;
}
.section {
    position: relative;
    margin-bottom: 20px;
}
.stream-container {
    text-align: center;
}
.stream-container img {
    width: 100%;
    height: 100%;
    border:1px solid #16191d;
    border-radius: 4px;
}
.table>tbody>tr>td, .table>tbody>tr>th, .table>tfoot>tr>td,
.table>tfoot>tr>th, .table>thead>tr>td, .table>thead>tr>th {
    border: 0;
    padding: 5px
}
.arrows {
    table-layout: fixed;
    margin-bottom: 0
}
.arrows tr:first-child td {
 padding-top: 0;
}
.arrows tr:last-child td {
    padding-bottom: 0;
}
.arrows .sides {
    padding-bottom: 0
}
.arrows .sides.left-side {
    padding-left: 0;
}
.arrows .sides.right-side {
    padding-right: 0;
}
.arrows .btn {
    width: 100%;
    background: #21252b;
```

```
    color: white;
    line-height: 50px;
    padding: 0;
    font-size: 2em;
    border: 0;
}
.arrows .btn:hover, .arrows .btn:focus, .arrows .btn.pushed {
    background: #16191d;
}
.arrows .sides .btn {
    line-height: 110px;
}
.console-wrapper .console {
    background: #16191d;
    color: #ffffff;
    width: 100%;
    height: 290px;
    border-radius: 4px;
    padding: 10px;
    text-align: left
}
.console-wrapper .console pre {
    background-color: inherit;
    color: inherit;
    border: 0;
    border-radius: 0;
    padding: 0;
    padding-right: 5px;
    margin: 0;
    width: 100%;
    max-height: 100%;
    overflow-x: hidden;
    overflow-y: auto;
    white-space: pre-wrap;       /* css-3 */
    white-space: -moz-pre-wrap;  /* Mozilla, since 1999 */
    white-space: -pre-wrap;      /* Opera 4-6 */
    white-space: -o-pre-wrap;    /* Opera 7 */
    word-wrap: break-word;
}
.console-wrapper .console pre::-webkit-scrollbar {
    background-color: #282c34;
    border-radius: 2px;
    width: 6px;
}
.console-wrapper .console pre::-webkit-scrollbar-thumb{
    background-color: #06090d;
    border-radius: 2px;
}
```

```
@media(min-width: 991px) {
    .console-wrapper.section {
        top: -130px;
    }
    .table-wrapper.section {
        bottom: -310px;
    }
    .stream-container {
        height: 420px;
    }
}
```

Javascript (Interface Behavior Layer)

```
/* Global Functions */
Object.compare = function (obj1, obj2) {
    //Loop through properties in object 1
    for (var p in obj1) {
        //Check property exists on both objects
        if (obj1.hasOwnProperty(p) !== obj2.hasOwnProperty(p)) return
false;

        switch (typeof (obj1[p])) {
            //Deep compare objects
            case 'object':
                if (!Object.compare(obj1[p], obj2[p])) return false;
                break;
            //Compare function code
            case 'function':
                if (typeof (obj2[p]) == 'undefined' || (p != 'compare'
&& obj1[p].toString() != obj2[p].toString())) return false;
                break;
            //Compare values
            default:
                if (obj1[p] != obj2[p]) return false;
        }
    }

    //Check object 2 for any extra properties
    for (var p in obj2) {
        if (typeof (obj1[p]) == 'undefined') return false;
    }
    return true;
};
/* Global Variables */
// Elements selectors
selectors = {
    console: ".console pre",
    stream: ".stream-container img",
    errorContainer: "#error_container",
    arrows: {
        general: ".arrows .btn",
        left: ".arrows #left-arrow",
        up: ".arrows #up-arrow",
        right: ".arrows #right-arrow",
        down: ".arrows #down-arrow",
        enter: ".arrows #recognize"
    }
};
// pressed buttons
```

```
pressedArrows = {
    left: 0,
    up: 0,
    right: 0,
    down: 0
};
recognizeKeyPressed = false;
// states
baseURL= window.location.protocol + "//" + window.location.host + "/";

/* Bind functions to events when document is ready */
$( document ).ready(function() {
    // Check if stream kicked off
    checkStream();
    //Bind keys to callbacks
    arrowKeysBinding({
        left: arrowKeyBehavior("left"),
        up: arrowKeyBehavior("up"),
        right: arrowKeyBehavior("right"),
        down: arrowKeyBehavior("down"),
        enter: {
            press: function() {
                if(!recognizeKeyPressed) {
                    recognizeKeyPressed = true;
                    $(selectors.arrows.enter).addClass("pushed");
                    recognize();
                }
            },
            release: function() {
                recognizeKeyPressed = false;
                $(selectors.arrows.enter).removeClass("pushed");
            }
        }
    });
    window.setInterval(function(){
        checkStream();
        synchronize();
    }, 1000);
});

/* Global Functions */

function writeToConsole(textToWrite) {
    var $console = $(selectors.console),
        startSymbol = '>',
        scrollBottomPosition = $console.prop("scrollHeight") -
$console.scrollTop() - $console.outerHeight();
    $console.append(startSymbol+" "+textToWrite+"\n");
```

```
    // scroll to bottom if the console is already at bottom
    if(scrollBottomPosition < 20) {
        $console.scrollTop($console.prop("scrollHeight"));
    }
    return true;
}

checkStream = (function(){
    var streamIsUp = false,
        img = new Image(),
        streamURL = $(selectors.stream).attr('data-src');

        img.onload = function() {
            if(!streamIsUp){
                streamIsUp = true;
                $(selectors.stream).attr('src', streamURL);
                writeToConsole('Stream is Ready!');
            }
        }
        img.onerror = function(){
            if(streamIsUp){
                streamIsUp = false;
                writeToConsole('Stream has Error!');
            }
        }
    return function() {
        img.src = streamURL;
        return true;
    }
})();

function arrowKeyBehavior(arrow) {
    return {
        press: function() {
            if(!pressedArrows[arrow]) {
                pressedArrows[arrow] = 1;
                $(selectors.arrows[arrow]).addClass("pushed");
                synchronize();
            }
        },
        release: function() {
            if(pressedArrows[arrow]) {
                pressedArrows[arrow] = 0;
                $(selectors.arrows[arrow]).removeClass("pushed");
                synchronize();
            }
        }
    };
```

```
}
// This functions sends current states to the server and ask for Robot
state to display on console

synchronize = (function () {
    var syncResponse = {
        waiting: false,
        hadError: false,
        state: ''
    };
    var moveInstructions = {};
    var oldmoveInstructions = {};
    var secondsCounter = 0;

    return function(){
        if(syncResponse.waiting) { return }

        secondsCounter++;
        moveInstructions = calculateMoveInstructions();
        if(secondsCounter < 4 && Object.compare(oldmoveInstructions,
moveInstructions)) {return}

        oldmoveInstructions = jQuery.extend({}, moveInstructions);
        if(secondsCounter <= 4) {
            secondsCounter=0;
        }

        syncResponse.waiting = true;
        $.ajax({
        method: "POST",
        url: baseURL + "move",
        data: moveInstructions
        })
        .done(function(result) {
            syncResponse.hadError = false;
            if(syncResponse.state != result.state) {
                syncResponse.state = result.state;
                writeToConsole(result.state);
            }
        })
        .fail(function(requestObject, error, errorThrown) {
            if(!syncResponse.hadError) {
                syncResponse.hadError = true;
                writeToConsole("An error happened! check browser
console for more info");

$(selectors.errorContainer).html(requestObject.responseText);
                console.log(requestObject);
```

```
                console.log(error);
                console.log(errorThrown);
            }
        })
        .always(function() {
            syncResponse.waiting = false;
        });
    }
})();

recognize = (function () {
    var recognizeResponse = {
        waiting: false
    };

    return function(){
        if(recognizeResponse.waiting) { return }

        recognizeResponse.waiting = true;
        $.ajax({
        method: "POST",
        url: baseURL + "recognize"
        })
        .done(function(result) {
            console.log(result);
            writeToConsole(result.state);
        })
        .fail(function(requestObject, error, errorThrown) {
            writeToConsole("An error while recognizing happened! check
browser console for more info");

$(selectors.errorContainer).html(requestObject.responseText);
            console.log(requestObject);
            console.log(error);
            console.log(errorThrown);
        })
        .always(function() {
            recognizeResponse.waiting = false;
        });
    }
})();

function calculateMoveInstructions() {
    var moveInstructions = jQuery.extend({}, pressedArrows);
    if(moveInstructions.up && moveInstructions.down) {
        moveInstructions.up = 0;
        moveInstructions.down = 0;
    }
```

```
    if(moveInstructions.left && moveInstructions.right) {
        moveInstructions.left = 0;
        moveInstructions.right = 0;
    }
    return moveInstructions;
}

function arrowKeysBinding(callback) {
    $(document).keydown(function(e) {
        switch(e.which) {
            case 13: // enter
                callback.enter.press();
                break;
            case 37: // left
                callback.left.press();
                break;

            case 38: // up
                callback.up.press();
                break;

            case 39: // right
                callback.right.press();
                break;

            case 40: // down
                callback.down.press();
                break;

            default: return; // exit this handler for other keys
        }
        e.preventDefault(); // prevent the default action (scroll /
move caret)
    });

    $(document).keyup(function(e) {
        switch(e.which) {
            case 13: // enter
                callback.enter.release();
                break;
            case 37: // left
                callback.left.release();
                break;

            case 38: // up
                callback.up.release();
                break;
```

```
            case 39: // right
                callback.right.release();
                break;

            case 40: // down
                callback.down.release();
                break;

            default: return; // exit this handler for other keys
        }
        e.preventDefault(); // prevent the default action (scroll /
move caret)
    });

    // Mouse click events
    $(selectors.arrows.general).mousedown(function(event) {
        var clickedArrow = $(this).attr("data-arrow");
        switch (event.which) {
            case 1: // left mouse button only
                callback[clickedArrow].press();
                break;
            default: return; // exit this handler for other keys
        }
    });

    $(selectors.arrows.general).mouseup(function(event) {
        var clickedArrow = $(this).attr("data-arrow");
        switch (event.which) {
            case 1: // left mouse button only
                callback[clickedArrow].release();
                break;
            default: return; // exit this handler for other keys
        }
    });
}
```

Flask Application (Server-side script)

```
#!/usr/bin/env python
from importlib import import_module
import json
import os
from flask import Flask, render_template, Response, request, url_for,
jsonify, g
# from camera.camera_opencv import Camera
# Raspberry Pi camera module (requires picamera package)
from camera.camera_pi import Camera
import smbus
import time
import sys
from PIL import Image
import picamera
import io
'''
camera = picamera.PiCamera()
camera.resolution = (300, 300)
camera.framerate = 10
'''
bus = smbus.SMBus(1)
app = Flask(__name__)
frame = 0

@app.route('/')
def index():
    """Video streaming home page."""
    return render_template('index.html')

def gen(camera):
    """Video streaming generator function."""
    global frame
    while True:
        global frame
        frame = camera.get_frame()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


@app.route('/video_feed')
def video_feed():
    """Video streaming route. Put this in the src attribute of an img
tag."""
    return Response(gen(Camera()),
```

```
                    mimetype='multipart/x-mixed-replace;
boundary=frame')

@app.route('/move', methods=['POST'])
def move_robot():
    state = ''

    move = {
        'left': int(request.form['left']),
        'up': int(request.form['up']),
        'right': int(request.form['right']),
        'down': int(request.form['down'])
    }
    if move['up'] and move['down']:
        move['up']=0
        move['down']=0

    if move['left'] and move['right']:
        move['left']=0
        move['right']=0

    # Move the car
    if move['up'] and move['right']:
        state="Moving: up-right"
        bus.write_byte_data(0x21, 0x00, 9)

    elif move['up'] and move['left']:
        state="Moving: up-left"
        bus.write_byte_data(0x21, 0x00, 7)

    elif move['down'] and move['right']:
        state="Moving: down-right"
        bus.write_byte_data(0x21, 0x00, 3)

    elif move['down'] and move['left']:
        state="Moving: down-left"
        bus.write_byte_data(0x21, 0x00, 1)

    elif move['up']:
        state="Moving: up"
        bus.write_byte_data(0x21, 0x00, 8)

    elif move['down']:
        state="Moving: down"
        bus.write_byte_data(0x21, 0x00, 2)

    elif move['left']:
        state="Moving: left"
```

```
        bus.write_byte_data(0x21, 0x00, 4)

    elif move['right']:
        state="Moving: right"
        bus.write_byte_data(0x21, 0x00, 6)

    else:
        state="Stopped"
        bus.write_byte_data(0x21, 0x00, 5)

    return jsonify({
        'state': state
    })

@app.route('/recognize', methods=['POST'])
def recognize_picture():
    global first

    state = 'Recognizing...'
    global frame
    image = Image.open(io.BytesIO(frame))
    image.save('/home/pi/Pics/2','jpeg')
    sys.stdout.write("qq")
    sys.stdout.flush()

    text = ''
    length = sys.stdin.read(4)
    sys.stdout.write("ZZzz")
    sys.stdout.flush()
    sys.stdout.write(str(length))
    sys.stdout.flush()
    rec = sys.stdin.read(int(length))
    state = rec

    return jsonify({
        'state': state
    })

if __name__ == '__main__':
    app.debug = False
    app.run(host='0.0.0.0', threaded=True, port=5000)
```

TensorFlow Image Classification Script

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
```

```
import threading
import sys
import time
import argparse
import os.path
import re
import sys
import tarfile
import threading
import numpy as np
from app import frame
import io
from six.moves import urllib
import tensorflow as tf
FLAGS = None
from PIL import Image
# pylint: disable=line-too-long
DATA_URL =
'http://download.tensorflow.org/models/image/imagenet/inception-2015-
12-05.tgz'
# pylint: enable=line-too-long
text = ''

class NodeLookup(object):

  def __init__(self,
               label_lookup_path=None,
               uid_lookup_path=None):
    if not label_lookup_path:
      label_lookup_path = os.path.join(
          FLAGS.model_dir,
'imagenet_2012_challenge_label_map_proto.pbtxt')
    if not uid_lookup_path:
      uid_lookup_path = os.path.join(
          FLAGS.model_dir, 'imagenet_synset_to_human_label_map.txt')
    self.node_lookup = self.load(label_lookup_path, uid_lookup_path)

  def load(self, label_lookup_path, uid_lookup_path):
    if not tf.gfile.Exists(uid_lookup_path):
      tf.logging.fatal('File does not exist %s', uid_lookup_path)
    if not tf.gfile.Exists(label_lookup_path):
      tf.logging.fatal('File does not exist %s', label_lookup_path)

    # Loads mapping from string UID to human-readable string
    proto_as_ascii_lines = tf.gfile.GFile(uid_lookup_path).readlines()
    uid_to_human = {}
    p = re.compile(r'[n\d]*[ \S,]*')
    for line in proto_as_ascii_lines:
```

```
      parsed_items = p.findall(line)
      uid = parsed_items[0]
      human_string = parsed_items[2]
      uid_to_human[uid] = human_string

    # Loads mapping from string UID to integer node ID.
    node_id_to_uid = {}
    proto_as_ascii = tf.gfile.GFile(label_lookup_path).readlines()
    for line in proto_as_ascii:
      if line.startswith('  target_class:'):
        target_class = int(line.split(': ')[1])
      if line.startswith('  target_class_string:'):
        target_class_string = line.split(': ')[1]
        node_id_to_uid[target_class] = target_class_string[1:-2]

    # Loads the final mapping of integer node ID to human-readable
string
    node_id_to_name = {}
    for key, val in node_id_to_uid.items():
      if val not in uid_to_human:
        tf.logging.fatal('Failed to locate: %s', val)
      name = uid_to_human[val]
      node_id_to_name[key] = name

    return node_id_to_name

  def id_to_string(self, node_id):
    if node_id not in self.node_lookup:
      return ''
    return self.node_lookup[node_id]

def create_graph():
  """Creates a graph from saved GraphDef file and returns a saver."""
  # Creates graph from saved graph_def.pb.
  with tf.gfile.FastGFile(os.path.join(
      FLAGS.model_dir, 'classify_image_graph_def.pb'), 'rb') as f:
    graph_def = tf.GraphDef()
    graph_def.ParseFromString(f.read())
    _ = tf.import_graph_def(graph_def, name='')


def run_inference_on_image(image):

  if not tf.gfile.Exists(image):
    tf.logging.fatal('File does not exist %s', image)
  image_data = tf.gfile.FastGFile(image, 'rb').read()

  # Creates graph from saved GraphDef.
```

```
  with tf.Session() as sess:

    softmax_tensor = sess.graph.get_tensor_by_name('softmax:0')
    predictions = sess.run(softmax_tensor,
                           {'DecodeJpeg/contents:0': image_data})
    predictions = np.squeeze(predictions)

    # Creates node ID --> English string lookup.
    node_lookup = NodeLookup()

    top_k = predictions.argsort()[-FLAGS.num_top_predictions:][::-1]
    global text
    for node_id in top_k:
      human_string = node_lookup.id_to_string(node_id)
      score = predictions[node_id]
      #print('%s (score = %.5f)' % (human_string, score))
      text = text + ('%s (score = %.5f)' % (human_string, score)) +
"\n"

def maybe_download_and_extract():
  """Download and extract model tar file."""
  dest_directory = "/home/pi/tar"
  if not os.path.exists(dest_directory):
    os.makedirs(dest_directory)
  filename = DATA_URL.split('/')[-1]
  filepath = os.path.join(dest_directory, filename)
  if not os.path.exists(filepath):
    def _progress(count, block_size, total_size):
      sys.stdout.write('\r>> Downloading %s %.1f%%' % (
          filename, float(count * block_size) / float(total_size) *
100.0))
      sys.stdout.flush()
    filepath, _ = urllib.request.urlretrieve(DATA_URL, filepath,
_progress)
    print()
    statinfo = os.stat(filepath)
    print('Successfully downloaded', filename, statinfo.st_size,
'bytes.')
  tarfile.open(filepath, 'r:gz').extractall(dest_directory)


#def main():

if __name__ == '__main__':

  parser = argparse.ArgumentParser()
```

```
parser.add_argument(
    '--model_dir',
    type=str,
    default='/home/pi/tar',
    help="""\
    Path to classify_image_graph_def.pb,
    imagenet_synset_to_human_label_map.txt, and
    imagenet_2012_challenge_label_map_proto.pbtxt.\
    """
)
parser.add_argument(
    '--image_file',
    type=str,
    default='',
    help='Absolute path to image file.'
)
parser.add_argument(
    '--num_top_predictions',
    type=int,
    default=5,
    help='Display this many predictions.'
)
FLAGS, unparsed = parser.parse_known_args()
#tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
#main()
maybe_download_and_extract()
#image = (FLAGS.image_file if FLAGS.image_file else
#         os.path.join(FLAGS.model_dir, 'cropped_panda.jpg'))
image = os.path.join('/home/pi/Pics', '1.jpg')
sys.stdout.write("Wait Some Minutes \n")
sys.stdout.flush()
create_graph()
#thread_1.start()
run_inference_on_image(image)
sys.stdout.write("Ready!!!!!!!! \n")
sys.stdout.flush()

#run_inference_on_image()
#sys.stdout.write("Ready!!!!!!!! \n")
#sys.stdout.flush()
image = os.path.join('/home/pi/Pics', '2')
global text
sys.stdout.write("{{")
sys.stdout.flush()
while True:
  text =''
  request = sys.stdin.read(2)
  if request == "re":
```

```
      run_inference_on_image(image)
      if len(text)%2 != 0:
        text = text + ' '
      text = str(len(text)) + " " + text
      sys.stdout.write(text)
      sys.stdout.flush()
      text = ''
      #sys.stdout.write("an")
      #sys.stdout.flush()
    x = 1
```

Pulse Width Modulation Driver

```
#include "PWM.h"

void PWM_init_OC0(void){
    TCCR0 = (1<<WGM00) | (1<<WGM01) | (1<<COM01) | (1<<CS01);//fast PWM
                                                            // Non-inverted
                                                            // Pre-scaler =
8
                                                            //About 490 HZ
    TCNT0 = 0;
    OCR0 = 0;
    DDRB |= (1<<PB3);
}

void PWM_init_OC2(void){
    TCCR2 = (1 << WGM20) | (1 << WGM21) | (1 << COM21) | (1 << CS21);
                                                            //fast PWM with
                                                            // Non-inverted
                                                            // Pre-scaler =
8
    TCNT2 = 0;
    OCR2 = 0;
    DDRD |= (1 << PD7);
}

void PWM_init_OC1A(void){
    TCCR1B = (1 << WGM12) | (1 << CS11); // presacler = 8
    TCCR1A = (1 << WGM10) | (1 << COM1A1) | (1 << COM1B1);
    TCNT1 = 0;
    DDRD |= (1 << PD5);

}

void PWM_init_OC1B(void){
    TCCR1B = (1 << WGM12) | (1 << CS11); // presacler = 8
    TCCR1A = (1 << WGM10) | (1 << COM1A1) | (1 << COM1B1);
    TCNT1 = 0;
    DDRD |= (1 << PD4);
}

void PWM_Duty_Cycle_OC1A(unsigned char Duty_Cycle){
    OCR1A = Duty_Cycle;
}

void PWM_Duty_Cycle_OC1B(unsigned char Duty_Cycle){
    OCR1B = Duty_Cycle;
}
```

```
void PWM_Duty_Cycle_OC0(unsigned char Duty_Cycle){
    OCR0 = Duty_Cycle;
}

void PWM_Duty_Cycle_OC2(unsigned char Duty_Cycle){
    OCR2 = Duty_Cycle;
}
```

Twin Wires Interface (I2C Protocol)

```
#include "TWI.h"
#include <avr/io.h>

/*****************************************************/
/*I2c initialization*/
void TWI_init(void){
    PORTC |= (1 << PC0);
    PORTC |= (1 << PC1);
#if TWI_SPEED == 100
    TWBR=32;   // to set the speed 100kb/s  ,,, Speed 400khz TWBR=2;

#elif TWI_SPEED == 400
    TWBR=2;   // to set the speed 100kb/s  ,,, Speed 400khz TWBR=2;
#endif

    TWSR=0;   // TWPS=0; //PreScaler 1:1
    TWCR=(1<<TWEN)|(1<<TWEA); // enable the TWI peripheral
}
/*****************************************************/
/*I2c initialization*/
void TWI_init_slave(unsigned char address){
    TWBR = 32;
    TWSR = 0;
    TWCR = (1 << TWEN); // enable the TWI peripheral
    TWAR = ( (address << 1) & (0xfe)); // address of the slave
    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
}
/*****************************************************/
/*I2c recieve*/
unsigned char TWI_recieve(){
    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA); // enable the TWI
peripheral
    while(!(TWCR & (1<<TWINT)) );
    return TWDR;
}
/*****************************************************/
/*I2c listen*/
void TWI_listen(){
    while(!(TWCR & (1<<TWINT)) );
}
/*****************************************************/
/*Device Address*/
void TWI_address(unsigned char address){
    TWAR = ((address<<1)&(0xfe));
}
/*****************************************************/
```

```
/*Sending the Address*/
void TWI_send_DeviceAddress_Read(unsigned char address){
    TWDR=((address<<1)|(0x01));//Address+ReadBit
    TWCR=(1<<TWEN)|(1<<TWINT)|(1<<TWEA);
    while((!(TWCR & (1<<TWINT)))&& ((((TWSR&0xf8)!=0x40))));
}
/*****************************************************/
void TWI_send_DeviceAddress_Write(unsigned char address){
    TWDR=((address<<1)&(0xfe));//Address+WriteBit
    TWCR=(1<<TWEN)|(1<<TWINT);
    while((!(TWCR & (1<<TWINT)))&&((TWSR&0xf8)!=0x18));
}
/*****************************************************/
/*Sending a byte of data*/
void TWI_send_data(unsigned char Data){
    TWDR=Data;
    TWCR=(1<<TWEN)|(1<<TWINT);
    while((!(TWCR & (1<<TWINT)))&&(
(((TWSR&0xf8)!=0x28))||(((TWSR&0xf8)!=0xB8))));
}
/*****************************************************/
/*Sending a stop bit*/
void TWI_stop(void){
    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);
}
/*****************************************************/
/*Receive with ACK*/
unsigned char  TWI_recieve_data_ACK(void){
    /*while the TWIF is zero (job not finished) & status register not
telling u that you received anything ....wait*/
    while(((!(TWCR &
(1<<TWINT))))&&((((TWSR&0xf8)!=0x80))||((TWSR&0xf8)!=0x50)));
    TWCR=(1<<TWEN)|(1<<TWINT)|(1<<TWEA);
    return TWDR;
}
/*****************************************************/
/*Receive with NACK*/
unsigned char  TWI_recieve_data_NACK(void)
{
    while(((!(TWCR &
(1<<TWINT))))&&((((TWSR&0xf8)!=0x88))||((TWSR&0xf8)!=0x58)));
    TWCR=(1<<TWEN)|(1<<TWINT);
    return TWDR;
}
/*****************************************************/
```

Main Motor Control AVR Code

```
#include <avr/io.h>
#include <util/delay.h>
#include "PWM.h"
#include "TWI.h"

#define SPEED 100

void MotorBR_forward(unsigned char speed);
void MotorBR_backward(unsigned char speed);
void MotorBL_forward(unsigned char speed);
void MotorBL_backward(unsigned char speed);

void MotorFR_forward(unsigned char speed);
void MotorFR_backward(unsigned char speed);
void MotorFL_forward(unsigned char speed);
void MotorFL_backward(unsigned char speed);

void Move_right(unsigned char speed);
void Move_left(unsigned char speed);
void Move_backward(unsigned char speed);
void Move_forward(unsigned char speed);

void Move_up_right(unsigned char speed);
void Move_up_left(unsigned char speed);
void Move_down_right(unsigned char speed);
void Move_down_left(unsigned char speed);

void Stop();

int main(){
    unsigned char data_TWI = 0;
    DDRD = 0xff;
    DDRB = 0xff;

    while(1){

        TWI_init_slave(33);
        TWI_listen();

        data_TWI = TWI_recieve_data_ACK();

        if(data_TWI == 8){
            Move_forward(SPEED);
        }
        else if(data_TWI == 6) {
            Move_right(SPEED);
```

```
        }
        else if(data_TWI == 5) {
            Stop();
        }
        else if(data_TWI == 4) {
            Move_left(SPEED);
        }
        else if(data_TWI == 2) {

            Move_backward(SPEED);
        }
        else if(data_TWI == 9) {
            Move_up_right(SPEED);
        }
        else if(data_TWI == 7) {
            Move_up_left(SPEED);
        }
        else if(data_TWI == 3) {
            Move_down_right(SPEED);
        }
        else if(data_TWI == 1) {
            Move_down_left(SPEED);
        }
        else{
            Stop();
        }
    }
    return 0;
}

void MotorBR_backward(unsigned char speed){
    DDRD |= (1 << PD3);
    PORTD &= ~(1 << PD3);
    PWM_init_OC0();
    PWM_Duty_Cycle_OC0(speed);
}

void MotorBR_forward(unsigned char speed){
    DDRD |= (1 << PD3);
    PORTD |= (1 << PD3);
    PWM_init_OC0();
    PWM_Duty_Cycle_OC0(speed);
}

void MotorBL_forward(unsigned char speed){
    DDRD |= (1 << PD2);
    PORTD &= ~(1 << PD2);
    PWM_init_OC1A();
```

```
    PWM_Duty_Cycle_OC1A(speed);
}

void MotorBL_backward(unsigned char speed){
    DDRD |= (1 << PD2);
    PORTD |= (1 << PD2);
    PWM_init_OC1A();
    PWM_Duty_Cycle_OC1A(speed);
}

void MotorFR_backward(unsigned char speed){
    DDRD |= (1 << PD1);
    PORTD &= ~(1 << PD1);
    PWM_init_OC1B();
    PWM_Duty_Cycle_OC1B(speed);
}

void MotorFR_forward(unsigned char speed){
    DDRD |= (1 << PD1);
    PORTD |= (1 << PD1);
    PWM_init_OC1B();
    PWM_Duty_Cycle_OC1B(speed);
}

void MotorFL_forward(unsigned char speed){
    DDRD |= (1 << PD0);
    PORTD &= ~(1 << PD0);
    PWM_init_OC2();
    PWM_Duty_Cycle_OC2(speed);
}

void MotorFL_backward(unsigned char speed){
    DDRD |= (1 << PD0);
    PORTD |= (1 << PD0);
    PWM_init_OC2();
    PWM_Duty_Cycle_OC2(speed);
}

void Move_forward(unsigned char speed){
    MotorBL_forward(speed);
    MotorBR_forward(speed);
    MotorFR_forward(speed);
    MotorFL_forward(speed);
}

void Move_backward(unsigned char speed){
    MotorBL_backward(speed);
    MotorBR_backward(speed);
```

```
    MotorFR_backward(speed);
    MotorFL_backward(speed);
}

void Move_right(unsigned char speed){
    MotorBL_forward(speed);
    MotorBR_backward(speed);
    MotorFR_backward(speed);
    MotorFL_forward(speed);
}

void Move_left(unsigned char speed){
    MotorBL_backward(speed);
    MotorBR_forward(speed);
    MotorFR_forward(speed);
    MotorFL_backward(speed);
}

void Stop(){
    TCCR2 = 0;
    TCCR0 = 0;
    TCCR1B = 0;
    TCCR1A = 0;
    PORTD &= ~( (1 << PD4) | (1 << PD5) | (1 << PD7) );
    PORTB &= ~(1 << PB3);
}

void Move_up_right(unsigned char speed){
    MotorBL_forward(speed + 30);
    MotorBR_forward(speed - 30);
    MotorFR_forward(speed- 30);
    MotorFL_forward(speed + 30);
}

void Move_up_left(unsigned char speed){
    MotorBL_forward(speed - 30);
    MotorBR_forward(speed + 30);
    MotorFR_forward(speed + 30);
    MotorFL_forward(speed - 30);
}

void Move_down_right(unsigned char speed){
    MotorBL_backward(speed + 30);
    MotorBR_backward(speed - 30);
    MotorFR_backward(speed- 30);
    MotorFL_backward(speed + 30);
}
```

```
void Move_down_left(unsigned char speed){
    MotorBL_backward(speed - 30);
    MotorBR_backward(speed + 30);
    MotorFR_backward(speed + 30);
    MotorFL_backward(speed - 30);
}
```

References

[1] https://blog.openai.com/openai-technical-goals/

By ILYA SUTSKEVER, GREG BROCKMAN, SAM ALTMAN& ELON MUSK.

[2] Lee Spector Cognitive Science, Hampshire College, Amherst, MA 01060, USA
Available online 7 November 2006

[3] Egmont-Petersen, M., de Ridder, D., Handels, H.: Image processing with neural net-worksa review. Pattern recognition 35(10), 2279–2301 (2002)

[4] Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., Culurciello, E.: Hardware accelerated convolutional neural networks for synthetic vision systems. In: Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium

[5] Hinton, G.: A practical guide to training restricted boltzmann machines.

[6] Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors.

[7] Ji, S., Xu, W., Yang, M., Yu, K.: 3d convolutional neural networks for human action recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on 35(1), 221–231 (2013)

[8] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Fei-Fei, L.: Large-scale video classification with convolutional neural networks. In: Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. pp. 1725–1732. IEEE

(2014)

[9] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convo-

lutional neural networks. In: Advances in neural information processing systems.

pp. 1097–1105 (2012)

[10] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel,

L.D.: Backpropagation applied to handwritten zip code recognition. Neural compu-

tation 1(4), 541–551 (1989)

[11] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to doc-

ument recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)

[12] Nebauer, C.: Evaluation of convolutional neural networks for visual recognition.

Neural Networks, IEEE Transactions on 9(4), 685–696 (1998)

[13] Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural net-

works applied to visual document analysis. In: null. p. 958. IEEE (2003)

(PDF) An Introduction to Convolutional Neural Networks. Available from: https://www.researchgate.net/publication/285164623_An_Introduction_to_Convolutional_Neural_Networks [accessed Jul 12 2018].

[14] Raspberry Pi Essentials – Jack Creasey.

[15] Raspberry Pi Architecture Essentials – Andrew K. Dennis.

[16] Exploring Raspberry Pi, Interfacing to Real World with Embedded Linux – Derek Molloy.

[17] Raspberry Pi User Guide – Ebon Upton, Gareth Halfacree.

[18] The AVR MicroController and Embedded Systems, using Assembly and C – Muhammad Ali Mazidi, Sarmad Naimi, Sephr Naimi.

[19] https://electrosome.com/i2c/