Mobile Robot Tank with GPU Assistance

By

Allen Chih Lun Pan

A thesis submitted to the faculty of graduate studies

Lakehead University

In partial fulfillment of the requirements for the degree of

Master of Science in Electrical Computer Engineering

Department of Eletrical Engineering

Lakehead University

October 25 2013

Acknowledgements

I am very grateful to my supervisor Dr. Xiao Ping Liu, who has the patience that allows me to complete my Master study and allows me to explore different solutions to a problem. He is a great professor with great vision on robotics, but most importantly I am grateful for him to take me under his wing. His trust and support encourages me to complete my thesis and venture into field in search of a solution to any problem. I sincerely thankful for Dr. Xiao Ping Liu's time and effort for the past 2 years for the completion of my thesis.

I would like to express my gratitude to my co-supervisor Dr. Kefu Liu for his knowledge input and to ensure quality of my mobile robot tank design.

This entire robot will not exist without the countless hours and assistance from Mr. Kailash Bhatia, whose expertise and knowledge in fabrication and manufacturing are most critical to the existence of this mobile tank robot.

I would also like to thank Lakehead University and stuffs for all this years of education and support. Without them I would never have such wonderful life changing opportunity.

I delicate my thesis to my loving wife "Savage Monkey" (Izabela), who's kid-savageness restored my confidence and helped me discover my capability and passion of creation. Without her I would have never consider doing Master degree.

To my daughter "I Help!" (Zahra) whose personality is embedded into my mobile robot tank, which explains the robot's complexity and difficulty to follow directions and also its stubbornness. With my daughter's "volunteer" enlisted help, I modeled this mobile robot tank's logic after her

To my son "Explorer 2.0" (Zephyr), this unstoppable little boy, inspires the more ruggedness build structure and curiosity of exploration of the unknown territories. My son's navigation and mapping capability is featured in the V3 version of my mobile robot tank.

To my parents, I am extremely grateful for all the support and encouragement, their love and care is the only reason I am where I am. Whenever I need technical support and programming advice, my father is always there for me, whenever I misses food from back home, my mother will ship those food to me, and for my sister who always provided me opportunity for fixing extra technical issues and difficulties.

Abstract

Robotic research has been costly and tremendously time consuming; this is due to the cost of sensors, motors, computational unit, physical construction, and fabrication. There are also a lot of man hours clocked in for algorithm design, software development, debugging and optimizing.

Robotic vision input usually consists of 2 or more color cameras to construct a 3D virtual space. Additional cameras can also be added to enrich the detailed virtual 3D environment, however, the computational complexity increases when more cameras are added. This is due to not only the additional processing power that is required running the software but also the complexity of stitching multiple cameras together to form a sensor.

Another method of creating the 3D virtual space is the utilization of range finder sensors. These types of sensors are usually relatively expensive and still require complex algorithms for calibration and correlation to real life distances. Sensing of a robot position can be facilitated by the addition of accelerometers and gyroscope sensors.

A significant robotic design is robot interaction. One type of interaction is through verbal exchange. Such interaction requires an audio input receiver and transmitter on the robot. In order to achieve acceptable recognitions, different types of audio receivers may be implemented and many receivers are required to be deployed. Depending on the environment, noise cancellation hardware and software may be needed to enhance the verbal interaction performance. In this thesis different audio sensors are evaluated and implemented with Microsoft Speech Platform.

Any robotic research requires a proper control algorithm and logic process. As a result, the majority of these control algorithms rely heavily on mathematics. High performance computational processing power is needed to process all the raw data in real-time. However, any high performance computation proportionally consumes more energy, so to conserve battery life on the robot, many robotic researchers implement remote computation by processing the raw data remotely. This implementation has one major drawback: in order to transmit raw data remotely, a robust communication infrastructure is needed. Without a robust communication the robot will suffers in failures due to the sheer amount of raw data in communication.

I am proposing a solution to the computation problem by harvesting the General Purpose Graphic Processing Unit (GPU)'s computational power for complex mathematical raw data processing. This approach utilizes many-cores parallelism for multithreading real-time computation with a minimum of 10x the computational flops of traditional Central Processing Unit (CPU). By shifting the computation on the GPU the entire computation will be done locally on the robot itself to eliminate the need of any external communication system for remote data processing. While the GPU is used to perform image processing for the robot, the CPU is allowed to dedicate all of its processing power to run the other functions of the robot.

Computer vision has been an interesting topic for a while; it utilizes complex mathematical techniques and algorithms in an attempt to achieve image processing in real-time. Open Source Computer Vision

Library (OpenCV) is a library consisting of pre-programed image processing functions for several different languages, developed by Intel. This library greatly reduces the amount of development time.

Microsoft Kinect for XBOX has all the sensors mentioned above in a single convenient package with first party SDK for software development. I perform the robotic implementation utilizing Microsoft Kinect for XBOX as the primary sensor, OpenCV for image processing functions and NVidia GPU to off-load complex mathematical raw data processing for the robot.

This thesis's objective is to develop and implement a low cost autonomous robot tank with Microsoft Kinect for XBOX as the primary sensor, a custom onboard Small Form Factor (SFF) High Performance Computer (HPC) with NVidia GPU assistance for the primary computation, and OpenCV library for image processing functions.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Autonomous robot is one of the Holy Grail in robotic research and the majority of autonomous robots are sensor based robots, which imply heavy utilization of sensors input and feedback. That being said, such a design usually requires quality sensors for raw information for capturing with a minimum amount of errors. The quality sensor also comes with hefty pricing; this is not economical and puts sensors at high risk during prototyping process.

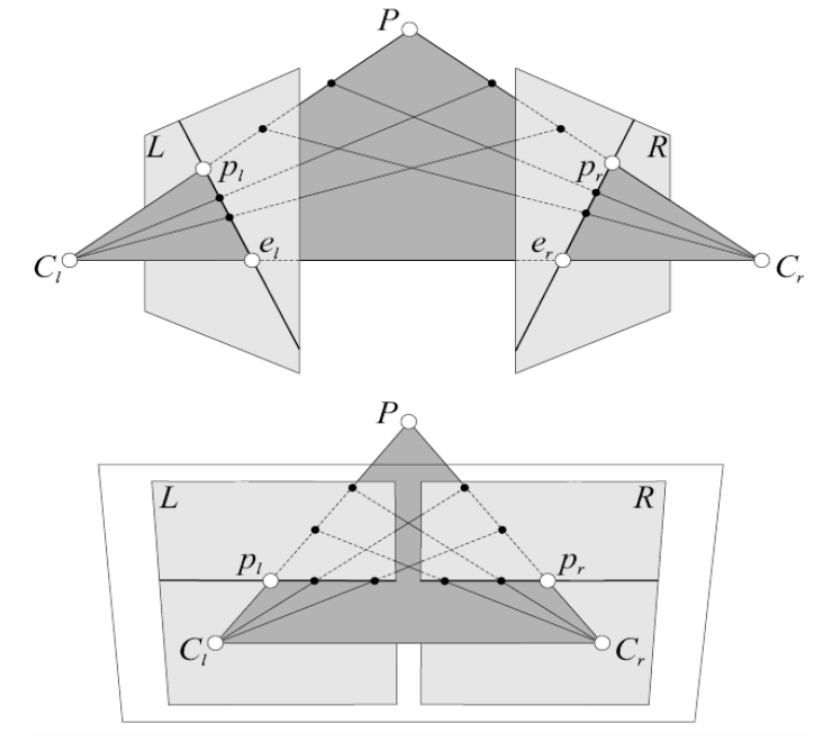
Interfacing several types of sensors is usually complicated as different sensors rely on different protocols and I/O. Moreover, the more advanced sensors also require proprietary software and API support; this is disastrous when combining different sensor providers and types in the same robot.

High accuracy robotics can be achieved by few means: either by well-tuned control algorithms, abundance of sensors for real-time error correction or strictly controlled environment. For both fine-tuned control algorithms and abundance of sensors, it requires a lot of computational power to achieve good performance in real-time.

1.1 Challenges in Stereo Computer Vision

The computer vision is defined as technology concerned with computational understanding and use of the information present in visual images [1]. Visual images are commonly considered as two dimensional representations of the real (3D) world. Computer stereo vision required acquisition of images with two or more cameras horizontally displaced from each other. In such a way, different views of a scene are recorded and could be computed for the needs of computer vision applications like the reconstruction of an original 3D scene. Stereo image pairs definitely contain more information about the real world than regular 2D images, for example scene depth, object contours, surface orientation and creases. [2].

The problem of matching the same points or areas in stereo image pairs is called the correspondence problem. Image pair matching could be performed by the algorithms that include the search and comparison of the parts of two images. If the camera geometry is known, the two dimensional search for corresponding points could be simplified to one dimensional search. This is done by rectification of images which is based on epipolar geometry (epipolar rectification) [3].



[4]

Figure 1. Epipolar rectification

Epipolar rectification is shown in Figure 1 [5]. If there are two pinhole cameras with the optical centers C_l and C_r, the scene point P will be projected onto the left (L) and right (R) image plane as p_l and pr. Any other possible scene point on a ray P-C_l will be projected on a line p_r-e_r which is called epipolar line of p_l, and where e_r is called epipole which geometrically represents the picture of left optical center C_l in the right camera. If the same is done on the left camera, rectification could be obtained by transforming the planes L and R in a way that p_l-e_l and p_r-e_r form a single line in the same plane. If all the scene points are transformed in the same way, the result will be that the same elements are situated on the same horizontal line in the left and right image [6], as shown in Figure 2, and the correspondence problem should be solved only in one (axis) direction:

$$x_r = x_l + d, \quad \text{while} \quad y_r = y_l$$

where d is the horizontal offset between the corresponding pixels (disparity), x_r and x_l represent the location of pixels on the horizontal axis in the right and left hand sides respectively, y_r and y_l the location of pixels on the vertical axis in the right and left hand sides.

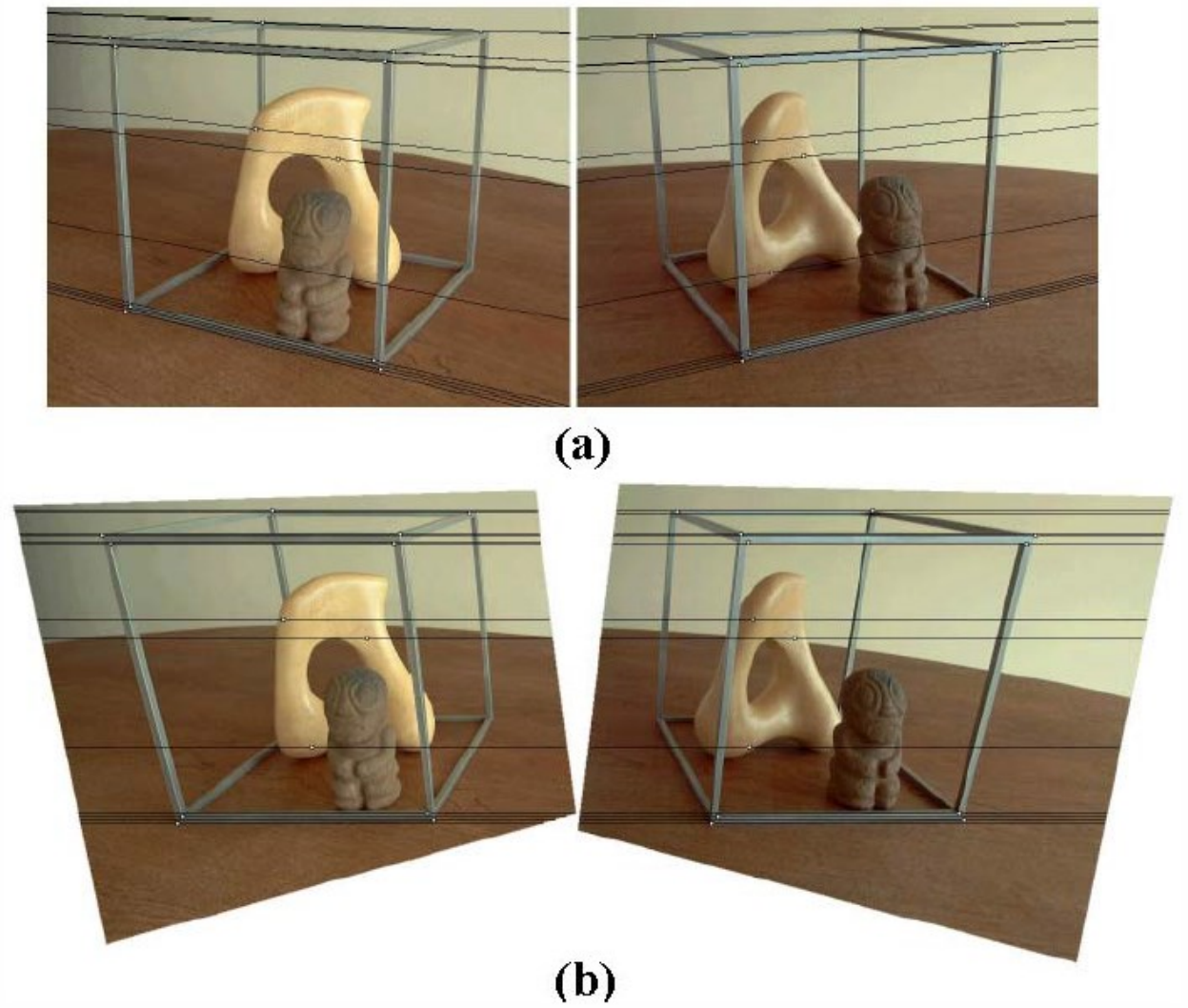


Figure 2. (a) Original image pair, (b) Rectified image pair [4]

With epipolar rectification we are assuming both cameras are parallel to each other and are identical (same focal lengths *f*). When the geometry of both camera systems is known, we can compute the distance [7]:

$$\frac{X}{Z} = \frac{x_l}{f} \qquad \text{and} \qquad \frac{X - B}{Z} = \frac{x_r}{f}$$

$$Z = \frac{B\,f}{x_l - x_r} = \frac{B\,f}{d}$$

where B is the distance between C_l and C_r, Z is the depth, f is the focal length, and X is the object's location to the right and left of the camera.

This approach requires two identical cameras with fixed position and angles; however, calibration is needed whenever there is a slight misalignment of the camera due to vibration or human error. The second problem is image error, as some pixel on one of cameras does not exist on the other camera. This corresponds to the situation wherein in some part of the scenes it may be visible to one camera but invisible to the other, which makes it impossible to calculate the depth of the object.

The other problem is with the assumption that cameras are assumed to be in a zero degree horizontal plane. This may not be the case when the robot is on an uneven terrain or over a hill. This would lead to inaccurate computation of depth distance.

Two cameras also increase the complexity and computation just for depth calculation, and as a result, the range finder has proven to be a much better alternative to depth information accumulation.

1.2 Challenges in Voice Recognitions

Voice command adds additional flexibility to the robot awareness and interaction, but also increases the complexity of the overall design. Over the decade speech recognition has improved over time, but it is far from being a solved problem.

Today's state-of-the-art systems are able to transcribe unrestricted continuous speech from broadcast data with acceptable performance. The advances arise from the increased accuracy and complexity of the models, which are closely related to the availability of large spoken and text corpora for training, and the wide availability of faster and cheaper computational means which have enabled the development and implementation of better training and decoding algorithms. Despite the extent of progress over the recent years, recognition accuracy is still extremely sensitive to the environmental conditions and speaking style. Channel quality, speaker characteristics and background noise have severe impact on the acoustic component of the speech recognizer, whereas the speaking style and the discourse domain have a large impact on the linguistic component. [8]

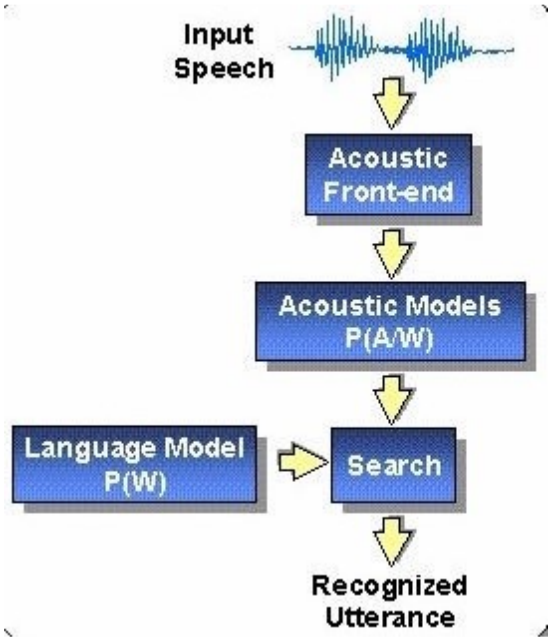


Figure 3. Basic model of speech recognition

There are 4 types of speech recognition: isolated words, connected words, continuous speech and spontaneous speech. Speech recognition, as modeled in Figure 3, can be achieved when a database of template pronunciation waveform (acoustic model) and grammatical rule sets have to be in place. This leads to one big problem: even without environmental noise, every region has a variation of how certain letters sound and there are some differences in rules. When an operator possesses an accent, the correct algorithm is needed to modify the template waveform to reduce errors.

Another problem is the database itself. To achieve considerable accurate results, the database grows overtime, which makes the search algorithm more important. The demand for intelligent robots evolves into the development of context aware speech recognition: Context-aware Speech to Text translator (COAST) [9].

1.3 Challenges in Cost and Performance

Rapid prototyping usually associates itself as a method of reducing development time, decreasing cost, increasing effective communication, and extending product life time [10]. Robot Tank is a small budget robotic project that employs the rapid prototyping technique to ensure a minimum amount of resources wasted.

The balance between in-house design/build and off-the-shelf product is crucial as each of them has its own advantage; the in-house build allows for full control, design and components selection of the final product, but is very time consuming and may end up more costly. On the other hand, off-the-shelf products provides well-tuned functionality as individual product, low cost of ownership as a massive

produce product, however, as an individual product it requires no development time, but to interface with other products may be time consuming.

With that in mind, the Robot Tank employed both approaches: in-house build and off-the-shelf product to maximize productivity and efficiency.

1.4 Contribution and Design Goal

The primary goal of the thesis project is to design a low cost autonomous robot. To ensure low cost, Microsoft Kinect for Xbox360 is utilized as primary navigation sensor for the robot.

The depth information from Kinect sensor has been used in [10] to achieve obstacle avoidance for a mobile robot. An indoor navigation with 3D mapping and reconstruction using Microsoft Kinect for Xbox360 has been developed in [11], which utilizes SLAM algorithm [11] to create a mapping of the environment and to reconstruct the complete sense in 3D with depth information.

In the above two Kinect Sensor systems, the client and host communication structure is used. The mobile robot relays the raw data to the host server, while the host server computes and gives directive back to the robot. This approach is to allow longer robot battery run time without burdening the robot with complex computation. It also ensures no computational lag in the real-time operation.

To achieve acceptable performance in real-time, computation of raw data is usually hosted on a computer or computer arrays. This creates massive upstream data flow to the host and very minimum downstream data to the robot because the downstream data is usually the directive or resultant of computation. This setup is very inconvenient and difficult for transportation and live demonstration.

GPU computation has exploded since 2006 by NVidia's Compute Unified Device Architecture (CUDA) for their Graphic Processor Unit (GPU). Research has shown that a single GPU can outperform CPU by 10X to 1000X depending on the testing condition [12], with comparison of CPU at a power envelope of 130 Watt TDP (Intel Core i7-960 [13]) and Nvidia's GPU at 182 Watt TDP (NVidia GeForce GTX 280 [14]) [15].

NVidia CUDA for robotics has been implemented for a robotic application: laser scanning for robotic vision with CUDA [16]. However, this paper only indicated that the implementation of CUDA is only on the laser scan process and no other computation, on the other hand, this paper did utilize Microsoft Kinect for Xbox360 for the laser scanner.

NVidia CUDA is also primarily used in image processing and object identification such as implementing Histogram of Oriented Gradients (HOG) on GPU [17], and is able to achieve speeds of up to 67X in comparison to CPU processing.

I propose to create a unique design by combining all of the above elements into one mobile robot tank. The Microsoft Kinect for XBOX360 is utilized as the primary sensor and NVidia CUDA is used for image processing to achieve high performance computation on board. The goal of this robot is to allow total autonomous navigation, voice command recognition, and object tracking.

1.5 Organization of Thesis

The organization of this thesis consists of 5 chapters:

- Chapter 1: Introduction to this thesis with challenges and objectives to be accomplished.
    - The various challenges in stereo computer vision, voice recognitions and cost/performance.
- Chapter 2: Different components that have been evaluated and implemented for the robot.
    - Different distance sensors evaluation and choice of implementation.
    - Different image sensors evaluation and choice of implementation.
    - Different acoustic sensors evaluation and choice of implementation.
    - Different x86 processors evaluation and choice of implementation.
    - Different general graphic processors' evaluation and choice of implementation.
    - Different mouse pointers evaluation and choice of implementation.
- Chapter 3: Robot's Software framework and design.
    - Overview of different libraries, Application Program Interfaces (APIs), and Software Development Kits (SDKs) that are used in the robot tank control software
    - Design of the robot tank control software and individual processing phases.
    - Design of the robot tank control hardware and implementations.
- Chapter 4: Performance enhancement and modification to achieve a better frame rate.
    - Raw data acquisition processing optimization.
    - Depth image processing revised and tweaking.
    - CPU threading.
    - GPU threading.
- Chapter 5: Future work, features and conclusion to this thesis project.
    - Scalability and portability of robot tank control software.
    - Processor architecture migration and code reusability.
    - The conclusion of the mobile robot tank project

Chapter 2 Components

2.1 Sensors Selection

Sensors are crucial to robotic design and as a result, several types of sensors have been considered and evaluated. The primary purposes of those sensors are to aid the robot for location awareness, object identification, directive interpolation, and situation alertness.

Based on the purpose of the sensors, I have broken it down to a few different categories: distance, visual, audio and orientation.

2.1.1 Distance Sensors

In order to determine the range or distance of a stationary object from the robot, I have evaluated a few types of available sensors: infrared (IR) sensor, ultra-sound/sonar (sound navigation and ranging) sensor and tachometers/encoders.

2.1.1.1 Infrared (IR) Sensor

Infrared light emits a longer wavelength than the visible light. The wavelength spectrum range from 700 nm to 1 mm which corresponds to frequency 400 THz to 300 GHz. Infrared light is essentially invisible to the human eye as it lies between microwave and visible light wave but with very minor reddish glow;, which is due to the end of visible light spectrum of 700 nm (Red), refer to **Table 1**.

There are five types of infrared Light Emit Diode (LED): near-infrared, short-wavelength infrared, mid-wavelength infrared, long-wavelength infrared and far infrared, see **Table 1**.

Table 1. Infrared types

Type	Abbreviation	Wavelength	Photon Energy
Near-infrared	NIR, IR-A DIN	700 – 1,400 nm	0.9 – 1.7 eV
Short-wavelength infrared	SWIR, IR-B DIN	1,400 – 3,000 nm	0.4 – 0.9 eV
Mid-wavelength infrared	MWIR, IR-C DIN	3,000 – 8,000 nm	0.15 – 0.04 eV
Long-wavelength infrared	LWIR, IR-C DIN	8,000 – 1,5000 nm	0.08 – 0.15 eV
Far infrared	FIR	15,000 – 100,000 nm	0.012 – 0.08 eV

Near-Infrared: Near-infrared is at the end of the red light spectrum. As a result, it has a minor reddish glow. It does not produce any heat and is heavily utilized in the medical community for both diagnostics and treatment.

Near-Infrared LEDs are usually produced with SiO_2 or Silicon Diode. Silicon Di-Oxide (SiO_2) is a low cost material and is abundant in the mass market. The near-Infrared has deep penetration and therefore is great for medical applications, such as Near-Infrared Spectroscopy (NIRS) for medical diagnosis of blood sugar level and pulse oximetry (blood sugar and pulse oximetry), pain relief and inflammatory relief.

Near-infrared light is also used as a primary choice for low light applications, being used in night vision goggles and surveillance cameras.

Short-Wavelength Infrared: More recent field application development has made good use of the short-wavelength infrared light. This is partially due to the development of Gallium Arsenide (InGaAs). The short-wavelength infrared can be made to be extremely sensitive to high light spectrum and allows better implementation for long-distance telecommunication, such as remote control and remote sensing.

Short-wavelength infrared lasers have been deployed for more advanced and accurate communication. There are many applications to the short-wavelength such as optical fiber communication system and free-space optical communication. The infrared light with a wavelength around 1,330 nm (least dispersion) or 1,550 nm (best transmission) are the best choices for standard silica fibers. [18]

Mid-Wavelength Infrared: Mid-wavelength infrared is primarily utilized in high temperature inspection for indoor applications, such as scientific research, laboratory equipment monitoring and indoor thermal imagining sensing. The temperature ranges from 300 degree Kelvin to 740 degree Kelvin or 26.85 °C to 466.85 °C [19].

There are multiple types of materials that can be constructed to form mid-wavelength Infrared: Indium Antimonide (InSb), Lead II Selenide (PbSe), p-Toluenesulfonyl Isocyanate (PtSi) and Mercury Cadmium Telluride (HgCdTe).

Long-Wavelength Infrared: This type of infrared is mostly well-known in military applications. It is commonly utilized in the ambient temperate environment, such as, heat-seeking weaponry, thermal night vision/goggles, and industrial inspection. The temperature ranges from 92.5 degree Kelvin to 400 degree Kelvin or -180.65 °C to 126.85 °C [19].

The material that is used to produce long-wavelength infrared is Mercury Cadmium Telluride (HgCdTe).

Far Infrared or Very-Long-Wavelength Infrared: This type of infrared is very sensitive to low temperatures: from 10.6 degree Kelvin to 140 degree Kelvin or -133.15 °C to *262.55 °C [19]. This extreme temperature range allows for the study of celestial bodies and space phenomena at far away galaxies.

2.1.1.2 Ultra-Sound and Sonar

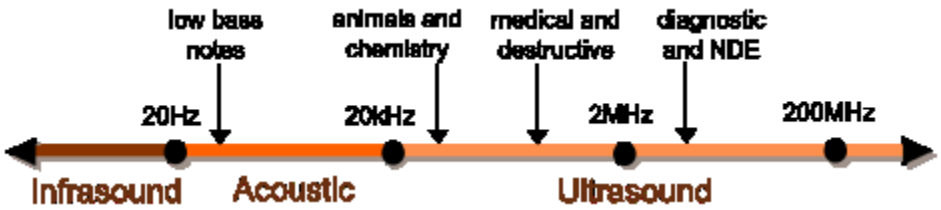


Figure 4. Sound wave frequency diagram[1]

[1] http://en.wikipedia.org/wiki/File:Ultrasound_range_diagram.svg

Ultra-sound and sonar both are very similar technologies that utilize the propagation of sound for detection and distance. The difference between ultrasound and sonar is its applications Sonar is usually referring to subterranean exploration such as underwater searching or natural resources discovery. On the other hand, ultrasound devices are used in many fields. Specifically, these sound waves can be used in ultrasonic imaging, which is a medical implementation for detecting invisible flows, and can also be used in non-contact cleaning or mixing to accelerate the molecular interactions in the chemical process.

Ultrasound/sonar waves oscillate at high frequency and usually operate at frequencies ranging between 20 kHz to several GHz range. A normal healthy human adult has an audible range from 20Hz to 20kHz, see Figure 4, and as a result, ultra-sound/sonar is inaudible to human hearing.

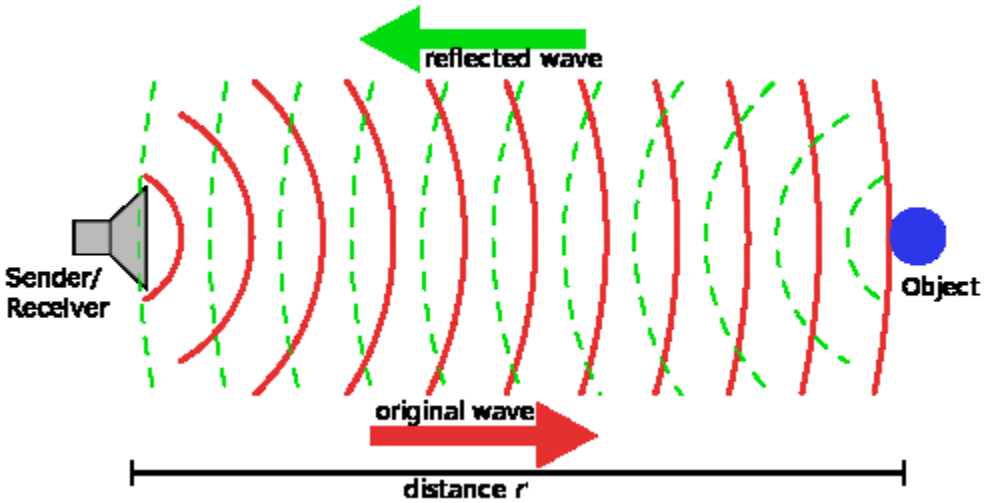


Figure 5. Principle of sonar/ultra-sound range finder[2]

One of the primary uses of ultra-sound is as a range finder. An ultrasound wave pulse is propagated in a particular direction and is partially reflected when an obstacle or object is in the path. This is also known as “echo”. The initial transmitted time and echo received time can be correlated to compute the approximate distance of the object or obstacle, see Figure 5.

However, since ultrasound and sonar are extremely sensitive to temperature, humidity, and other high frequency noise, recalibration may be needed to achieve high accuracy in proper distance detection.

[2] http://en.wikipedia.org/wiki/File:Sonar_Principle_EN.svg

2.1.1.3 Laser Rangefinder

Laser rangefinder utilizes a laser beam to determine the distance to the object. It has a similar principle as ultra-sound/sonar by measuring the time taken for a laser pulse to travel to the object and reflected back to the sender.

Laser rangefinder provides much higher accuracy than sonar/ultrasound, with the advantage of the speed of the light propagation. It has accuracy up to millimeters [20]). The laser pulse can be encoded to prevent jamming, reduce interference and improve overall accuracy even further.

2.1.1.4 Conclusion and Decision of Distance Sensor

All of the 3 types of sensors have their own advantages. Infrared sensors work in both dark and ambient environments; however they require separate emitter and receiver devices. This also has to do with the beam angle that was produced from the IR LED. Some IR LED have only 30 degree angle but long penetration, while some haave 120 degree angle with very short range. The other disadvantage is that the wavelength of both emitter and receiver must match each other, or else no detection will be determined by the receiver.

Ultrasound is great as a close proximity sensor; however, it can easily become problematic with the environmental noise and machine vibration. Most of field application test data shows that ultrasound for long range detection is not advisable; however, for close proximity distance approximation ultrasound has proven to be extremely cost effective. A similar technique has been implemented in the automobile industry for preemptive collision detection and parking assistance. As a result, I primarily utilize ultra-sound sensors with IR sensors as a backup to ensure proper detection and minimal false-positive errors.

Since both IR and ultra-sound will only be used in close proximity range, near-infrared sensors are the best choice due to the lowest cost and minor red glow, which ensures the ease of trouble shooting during the debugging phase.

As for getting the proper distance, the laser range finder is not cost effective, due to shear pricing ($9000.00 on Swiss Ranger SR400. On the other hand, Microsoft Kinect for XBOX360's built in laser range finder can obtain approximately 14400 lines per second for the MRSP (Manufacture Recommended Sale Price) cost of $149.99. That is also another reason why Microsoft Kinect for XBOX360 has been chosen as the primary sensor.

2.1.2 Imaging Sensor

Object following is an important task of mobile robots. This functionality requires some sort of data acquisition and identification and as a result, we evaluated two different types of digital color image sensors.

2.1.2.1 CCD Sensor

One type of digital color image sensor is a Charge-Coupled Device (CCD) sensor. CCD is technically an analog device: as incoming light strikes the material and is converted into a small electrical charge in the sensor. The charge is converted into a voltage one pixel at time as the system is reading from the chip.

This design requires an additional voltage conversion circuit to convert the voltage into digital information.

This linear reading and conversion greatly reduces the capturing speed. In exchange, it does provide much better image color and can provide much higher resolution. As a result, CCD sensor is usually more expensive to produce when compared to CMOS.

2.1.2.2 CMOS Sensor

CCD sensors are usually more expensive and have higher power consumption due to the need of additional voltage circuitry to convert analog data into digital voltage signal. This additional voltage circuitry also leads to bigger component packages. However, due to the demand of low cost, low power and compact electronics, CMOS sensors address such needs.

An Active-Pixel Sensor (APS) is an integrated circuit with an array of pixel sensors, which consists of a photo-detector and an active amplifier. General speaking, the CMOS image sensor consumes less power, is cheaper to manufacture and is immune to color bleeding found in CCD sensors. However, mainstream CMOS image sensors have a "rolling shutter" effect, which skews the image based on the movement direction of the camera or object.

2.1.2.3 Conclusion and Decision of Imaging Sensor

During the testing phase I have acquired a few webcams from different vendors for testing:

Microsoft's Webcam:

- Microsoft LifeCam Studio (CCD).
- Microsoft LifeCam Cinema (CCD).
- Microsoft LifeCam VX-5000 (CMOS).
- Microsoft Kinect for XBOX360 (CMOS).

Logitech's Webcam:

- Logitech Quickcam Pro 9000 (CMOS).
- Logitech Quickcam Communicate STX (CMOS).
- Logitech Orbit (CCD).

Apple's Webcam:

- iSight Firewire (CCD).
- iMac's Build-in iSight (CMOS).

Other Webcam:

- Lenovo T400 build-in Webcam (CMOS).
- No-brand 1.3 Mpixels Webcam (CMOS).

Both Microsoft LifeCam Studio and Cinema provide the best quality in both low light and illuminated environments and are capable of live video streaming at 1080P (LifeCam Studio) and 720P (LifeCam Cinema). However, both LifeCam devices share the same design and only differ by the image sensor (8 Mpixel / 5 Mpixels). The power requirements are identical, strict and extremely demanding; both require the entire 500 mA of current available from the USB hub, from which two USB ports are connected and shared. When other devices are connected to the hub, this had led to some problems.

Logitech's legacy webcam products utilize high quality CCD imaging sensor, however, the present product line-up has discontinued CCDs to utilize the cheaper CMOS imaging sensor. The overall quality between the Quickcam Pro 9000 and Orbit is comparable, but during low light environment it tends to pick up noise. The advantage of the Logitech CMOS webcam is very lax on the power requirement as both Logitech CMOS webcams only consume up to 250 mA per device.

Apple's Webcam, on the other hand, cannot distinguish the quality difference between both iSight implementations; both provide equally crystal clear image in both low light and well-lit environments due to the 2nd generation CMOS sensor that has been utilized in the iMac's iSight. The only problem for the iSight is that both models require much higher CPU resources during video capturing. Apple's iSight for Windows driver might contribute to the extra consumption of resources. The last difference between the two is that the Firewire version consists of auto focus and better aperture, while the iMac version does not.

Lenovo T400 built-in Webcam and the no-brand webcam are both noisy during low light and experience frame drop during low light capturing. However, the power consumption is quite low; at an average of 150 mA.

When it comes to quality, CCDs provide much better quality at the cost of power consumptions, while CMOS variants have no quality advantage but the simplicity and on-die silicon design transfers the savings to power. Depending on the application and situation both imaging sensors shine. In this robot tank I utilize Microsoft Kinect for XBOX 360's color camera as a primary image sensor, due to the power savings, since the raw data is going to be processed internally, the quality of image is not crucial, and any noise can be easily removed by a simple filter. Microsoft LifeCam Studio has also been implemented on the robot for recording purposes only. To save both resources and power, it will only be active for a specific time frame and situation.

2.1.3 Acoustic Sound Sensor

For voice command or environment audible noise recognition and filtering, some sort of analog to electrical signal sensor is needed. This type of sensor is also known as an acoustic-to-electric transducer or sensor. One of the most commonly used acoustic sound sensors is the "microphone". There are 3 types of the most commonly received patterns of microphone: omnidirectional, unidirectional, and bi-directional, see Figure 6.

Omnidirectional Pattern Unidirectional Pattern Bi-directional Pattern

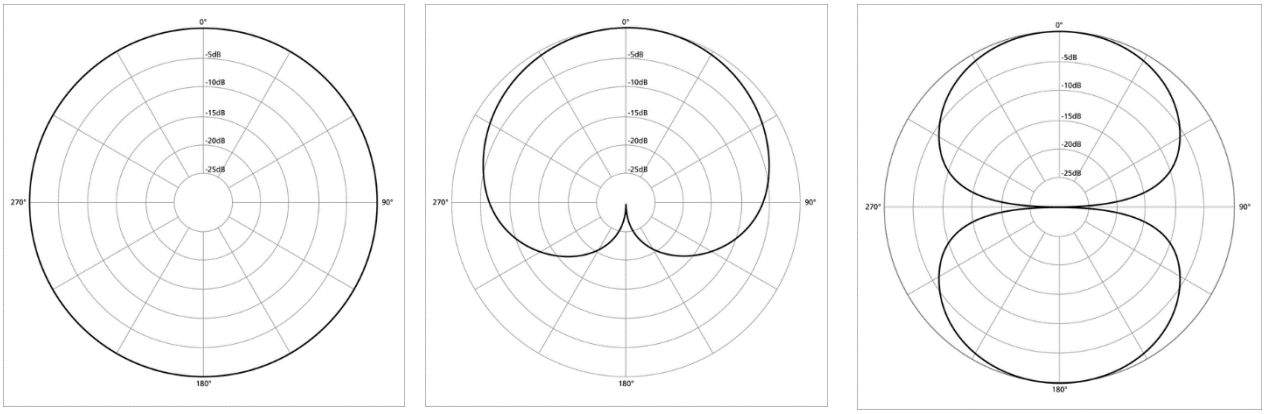


Figure 6. Acoustic sensor patterns

2.1.3.1 Omnidirectional Microphone

Omnidirectional microphones or non-directional microphones provide the best spherical sensitivity. However, in real life application the sensitivity of microphones is not good when the audio source is from the rear because the body of microphone usually blocks and disperses the sound when coming from behind.

2.1.3.2 Unidirectional Microphone

Like the name implies, the unidirectional microphone provides sensitivity to only 1 direction. This has the advantage of removing noise that is not in the direction of interest. On the other hand, this can also be a disadvantage, if the source of audio has moved out of the region/cone of sensitivity.

The most common unidirectional microphone is a “Cardioid” microphone as it has the sensitivity pattern of a cardioid.

2.1.3.3 Bi-directional Microphone

The bi-directional microphone has two sensitivity cones in the shape of a Figure 6, as the sensitivity is front-and-back. This allows better audio acquisitions for both sides of microphone. Some other directional microphones have a little bit different variation of polarization patterns to fit the applications’ needs.

2.1.3.4 Conclusion and Decision of Acoustic Sound Sensor

Now-a-days all the webcams have built-in microphones for seamless video and audio streaming. However, due to cost constraints, most utilize simple omnidirectional microphones and to achieve a more diverse range, multiple omnidirectional microphones are integrated into an array. This is the case in the Microsoft Kinect for XBOX360 and Playstation Eye.

Unidirectional microphones are a very bad choice as an acoustic sound sensor for robotics, as the odds of voice that have been issued could be from any direction.

There is a solution to the directional issue by pointing the onmi-directional microphone downward. This position prevents the shaft of microphone from blocking the incoming sound and by placing multiple of omni-direction microphones in an array also allows diversity in range, adding the capability for directional detection for incoming sound waves.

After testing different microphone setups, I decided to use the Microsoft Kinect for XBOX 360. This device has an omnidirectional microphone array and also positions the entire microphone array downward to eliminate shaft blocking.

Microsoft Kinect's microphone array has 4 microphones. The 1st microphone is placed 149 mm away from the 2nd microphone and the other 3 are 36mm, 40mm, and 37 mm apart, as shown in Figure 7. This configuration allows Microsoft Kinect to determine the sound source's direction based on the delay of sound. The delay of the 1st (left most) and 2nd microphone is up to 7 samples, while the 2nd, 3rd, and 4th have a minimum delay with each other.

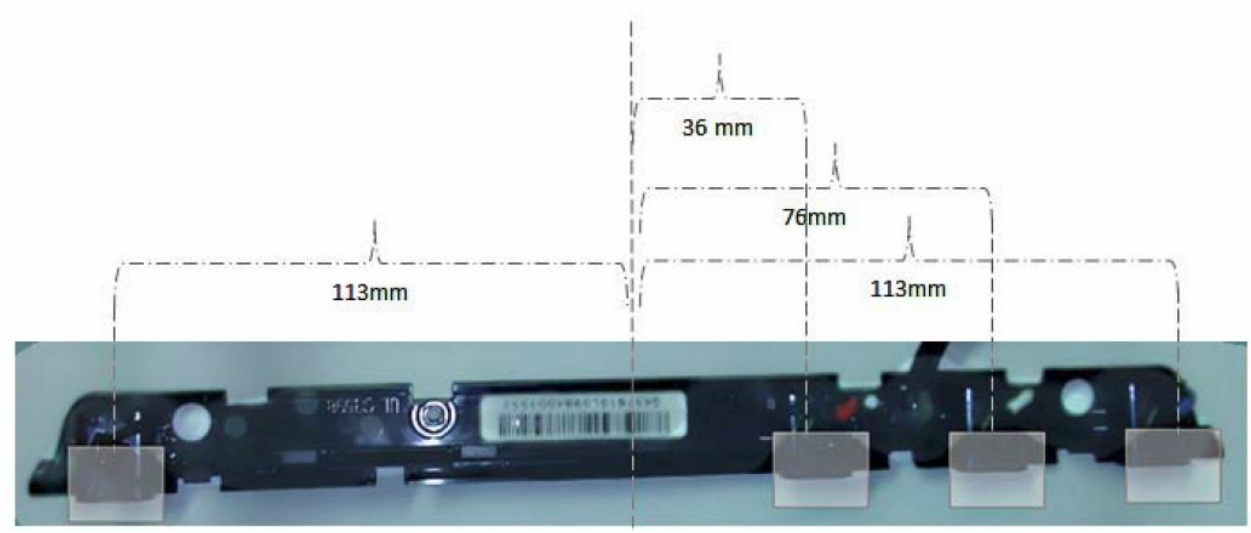


Figure 7. Microsoft Kinect microphone array

2.2 Processor Selection

This robot tank contains 4 types of processor: x86 Central Processor Unit (CPU), Ti Micro-Controller Unit (MCU)/ Digital Signal Processor (DSP), ARMs System-on-Chip (SoC), and General Purpose Graphic Processor Unit (GPGU).

In order to reuse previously established code, the onboard computer must be in the x86 architecture. As a result, there are only 3 major players in the market that I considered and evaluated. However, in order to evaluate their ability and suitability for the robot, the following factor of merit have been set: Thermal Design Power (TDP), Giga-Flops Performance (GFLOPS), Software Development Kit Support (SDK), and Single Instruction Multiple Data sets support (SIMD) and cost of ownership (Dollars).

To have a fair comparison and reduce overall power consumption I decided to limit my x86 CPU choices to mobile, low voltage, or ultra-low voltage processor choices and this has to be at least dual-core. Refer to **Table 2** for complete white paper comparison.

2.2.1 Intel x86 Families

There are 3 families of x86 CPU from Intel that I considered and evaluated, there are: Intel ATOM family, Intel Core 2 family, and Intel iCore Series family.

Intel ATOM Family: This is Intel's low budget and ultra-low-voltage x86 processor. Intel ATOM processor has an in-order instruction scheduler. This implies that instructions are fed into the instruction pipeline in exactly the order as they are fed to it by the binary code. No instruction re-ordering is done at the processor level. As a result the processor is considerably more sensitive to instruction latencies and dependency stalls caused by poor instruction scheduling by the compiler of your choice. Intel ATOM family performance chart is shown in Figure 8 [21].

Intel ATOM 330 "Diamondville": This is a 1st generation Intel ATOM processor operating at a frequency of 1.6GHz. It features dual-core with Intel hyper-threading technology, effectively 4 logic threads. The processor's operational TDP is 8 Watt. However, the first generation of Intel ATOM processor requires two chipsets in order to function: Intel 945GC (Northbridge) at 22.2 Watt and Intel ICH7 (Southbridge) at 3.33 Watt, which brings the total power consumption of Intel ATOM 330 platform to approximately 35 Watts.

ATOM 330 is a dual N270 in a single PBGA package, so its theoretical performance of Intel ATOM 330 is 6.4 GFLOPS [22].

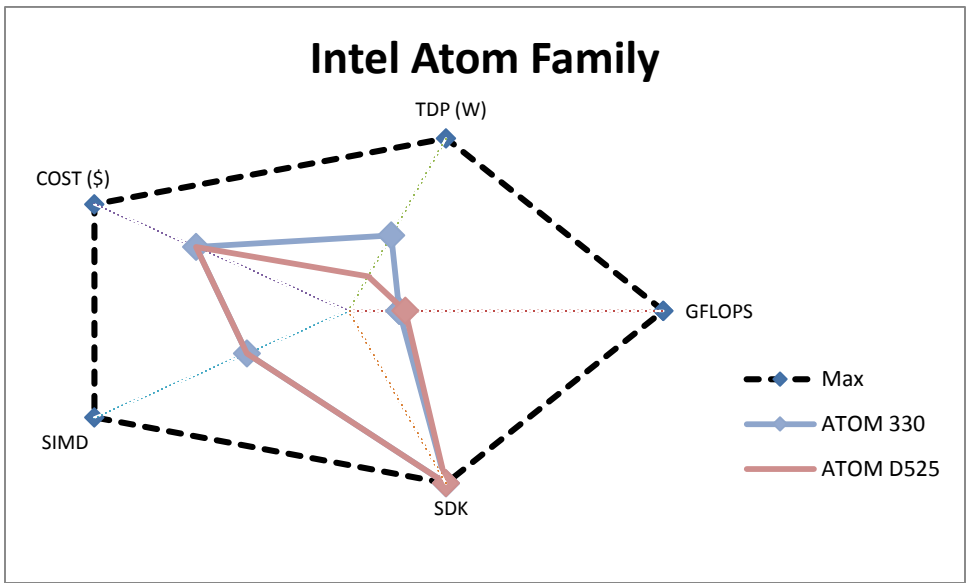


Figure 8. Intel ATOM family performance chart

Intel ATOM D525 "Pineview": This is a 2nd generation Intel ATOM processor operating at a frequency of 1.83GHz. It features dual-core with Intel hyper-threading technology, effectively 4 logic threads. The processor's operational TDP is 13 Watt. The downfall of the first generation ATOM is one additional chipset which consumes up to 4 times more power than the processor itself. The 2nd generation ATOM integrates the Northbridge into the processor itself and the only one additional chipset that is needed is

the Southbridge: Intel PCH NM10 at 2.1 Watt. Therefore, its total power consumption is approximately 16 Watts.

The theoretical processor performance of Intel ATOM D525 is 7.2 GFLOPS. [23]

Intel Core 2 Duo Family: This is a successor of Intel Pentium micro-architecture, a great departure to the Netburst micro-architecture as Netburst focuses on clock rate over thermal/power consumptions, while the core family mainly focuses on efficiency. As a result, the core family allows much longer battery life with rapid responses. Its performance is shown in Figure 9.

Intel Core 2 Duo Mobile T9550 "Penryn": Penryn is a refined or die shrink version of Intel Core 2 Duo architecture, also known in Intel process model as a "Tick". T9550 is a dual core processor and operating at 2.67GHz with TDP of 35 Watts. Just like most x86 processors, it also requires extra chipsets. There are various chipsets that can be paired up with Intel Penryn processor: Intel 965 series, Intel 3x/4x series and NVidia nForce.

The evaluation platform for the T9550 Core 2 Duo mobile processor is with Intel GM45 (Northbridge) at 12 Watt, and Intel ICH10R at 4.5 Watt, total of approximate 52 Watt.

The theoretical processor performance of Intel Core 2 Duo Mobile T9550 is 21.328 GFLOPS. [24]

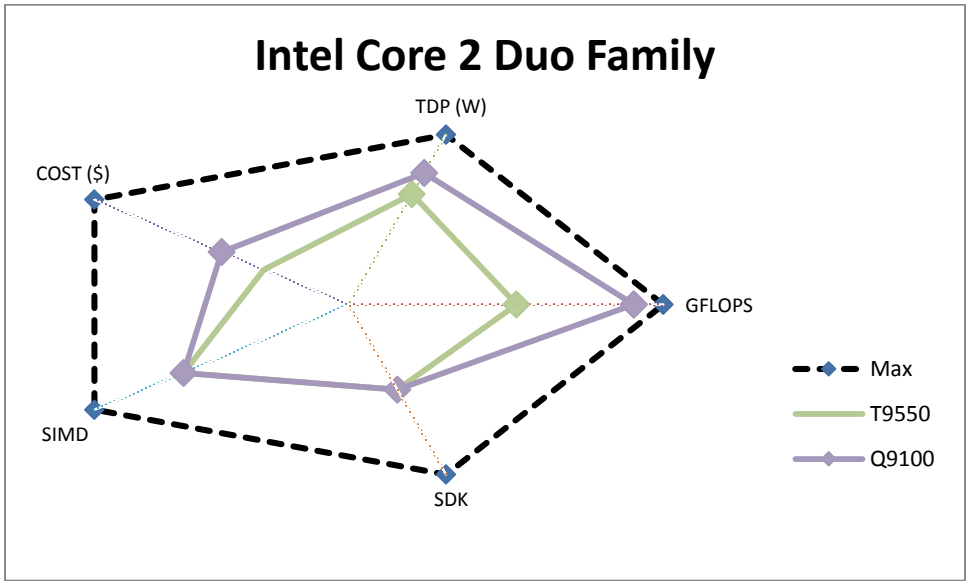


Figure 9. Intel Core 2 Duo family performance chart

Intel Core 2 Quad Mobile Q9100 "Penryn QC": The Q9100 is a quad-core variance of Core 2 duo mobile with double amount of level 2 cache. Its TDP slightly increases to 45 Watt and its clock rate reduces to 2.26 GHz. However, since it is quad core, it retains all the micro-architecture advantage of the dual core variance but with total of 4 logic threads. With the same platform as the T9550, the total power consumption increases to approximately 62 Watts.

The theoretical processor performance of Intel Core 2 Duo Mobile Q9100 is 36.256 GFLOPS. [25]

Intel iCore Family: This is a successor to the Core/Core 2 micro-architecture, known as "Nehalem". The main concept of iCore family is to further increase energy-efficiency and performance and as a result, Intel reintroduces hyper-threading technology" which allows 2 logic threads per core and integration of Northbridge (memory controller, PCI-express lanes, and Intel Graphic Processor (iGP)) into a single package. The performance of Intel iCore family is shown in Figure 10.

Intel Core i3-390M/i5-560M/i7-620M "Arrandale": The 1st generation Intel iCore mobile processor is essentially the same with very minor differences in features. As a result of this, at similar clock rates their performance level is at par with each other, however, the differences are as follows: i7 has higher Turbo Boost and Intel VT-x VM support, i5 has lower Turbo Boost and Intel VT-x VM Support, and i3 has none of the above features. [26]

Intel Core i3-390M/i5-560M/i7-620M are clocked at 2.67 GHz, Dual-core, with hyper-threading (4 Logic threadings). The TDP is 35 Watt and at 3.5 Watt Idle.

Intel Turbo Boost allows dynamic over-clocking when the application is in single treading mode and abundance of resources. This has not much of benefit for the robotic application, as most of the time the processor will be busy and the code will be in multi-threading mode.

Intel VT-x is hardware acceleration for Virtual Machine (VM) and has no benefit to this robotic application. Since this platform is no longer in need of Northbridge, the power consumption is also much lower. The evaluation platform consists of Intel QM57 with the total power consumption of approximate 39 Watt.

The theoretical processor performance of Intel Core i3-390M, i5-560M, and i7-620M is 21.328 GFLOPS [27], 21.328 GFLOPS [28], and 21.328 GFLOPS [29], respectively.

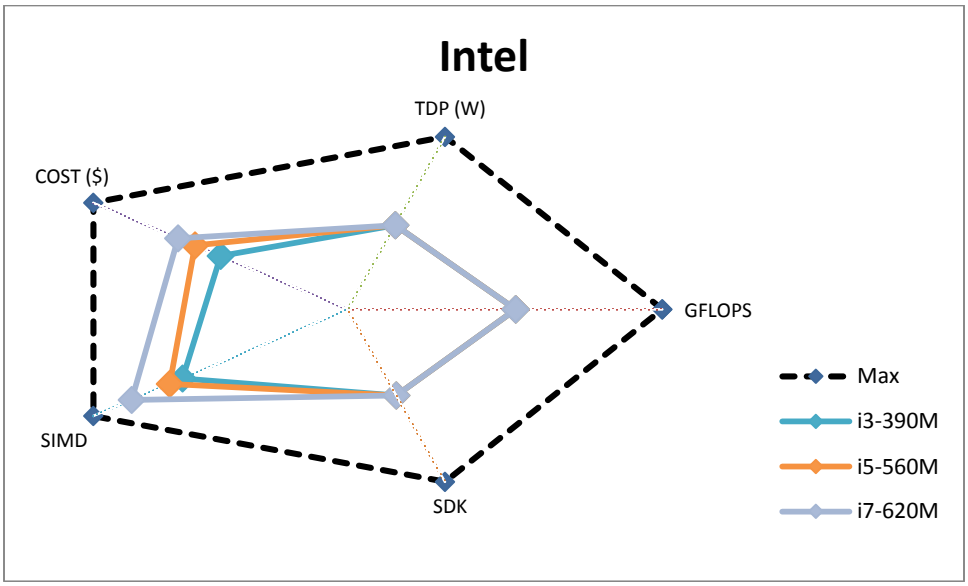


Figure 10. Intel Core-i family performance chart

2.2.1 AMD x86 Families

When energy efficiency and battery life, AMD is far from competitive. AMDs does not have good power/performance ratio. As a result, in the mobile sector, the majority of Original Equipment Manufacturer (OEM) and Original Design Manufacturer (ODM) still prefer Intel over AMD.

Since the acquisition of ATI, AMD is becoming more competitive in the mobile and low power areas. One of their primary arsenals is the Accelerated Processing Unit (APU) family, which is a combination of CPU with GPU on a single die. The performance of the APU family is shown in Figure 11.

AMD Fusion APU Family: The unification of CPU and GPU in a single die was announced by AMD back in the 2006 with acquisition of ATI. The idea was simple to combine the high rendering performance of ATI Radeon graphics and the raw computation performance of AMD processor into a single package and to reduce the total number of chip implementations down to two. The ATI Radeon graphic processor is extremely competitive in the 3D acceleration and gaming market, and so, AMD is attempting to leverage

this advantage to drive OEM/ODM away from Intel and towards AMD, as OEM/ODM no longer needs to purchase separate dedicated graphic chips to achieve good 3D performance.

AMD A8-3550MX "Llano": This is a 1st generation AMD APU. The CPU is the 2nd generation AMD K10 start micro-architecture which was introduced in 2010, and the GPU is ATI HD6000 BeaverCreek micro-architecture which was also introduced in 2010.

AMD A8-3550MX is a quad core APU with operational frequency of 2.0 GHz and TDP of 45 Watt. Since the Northbridge is also integrated to the APU, the only additional chip that is needed is one Southbridge: AMD A60M at 4.7 Watt, which brings the total power consumption for this evaluation platform to approximately 50 Watts.

The theoretical combined performance of AMD A8-3550MX is approximately 27 GFLOPS. [30]

AMD E-350 "Brazos": This is also a 1st generation APU but with more energy-efficiency in mind. AMD E-350 is a dual core processor with an operational frequency of 1.6 GHz and the TDP of 18 Watt. Just like any other APU it only needs one Southbridge: AMD A50M at 5.9 Watt, which brings the total power consumption for this evaluation platform to approximately 24 Watts.

The theoretical combined performance of AMD E-350 is approximately 7.4 GFLOPS. [30]

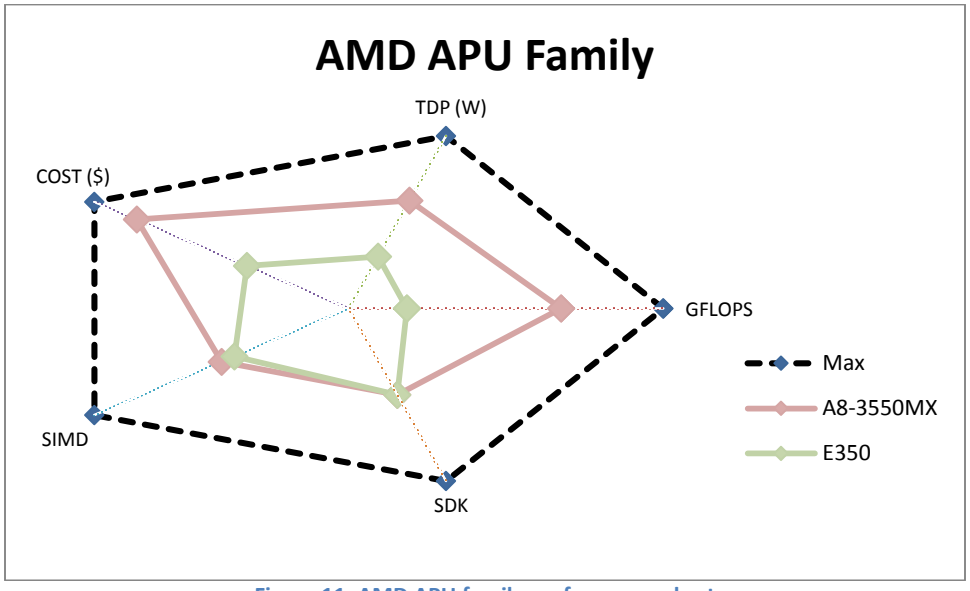


Figure 11. AMD APU family performance chart

2.2.1 VIA x86 Families

When it comes to low power, small footprint embedded systems, VIA is an old time veteran in this field. VIA Technologies design and manufacture a wide range of integrated circuits and complete embedded systems with VIA's own x86 processor or VIA's ARM SoC.

VIA Nano Family [31]: VIA Nano is VIA's 8th generation x86 CPU targeted for low power application, consumer and embedded system market. It features complete ground up x86 in-house design and

supports all the modern Intel x86 counter part's instruction sets and features (x86-64, SSE4, x86 VT, ECC…etc).

Via Nano is an out-of-order processor, which greatly increases general computation performance without the need of a specialized compiler. VIA claims 30% boost in performance in comparison to Intel's ATOM processor which has 50% higher clock rate [32]. VIA Nano's energy-efficacy is also another great strength, as VIA Nano is able to achieve the same TDP as Intel ATOM with greater performance, as shown in Figure 12.

VIA Eden X2 U4200 "Isaiah": This is a dual core variation of VIA Nano micro-architecture and features 1.0 GHz with TDP of only 6 Watt. Just like all the modern processors, the only additional chip that is required is the Southbridge: VIA VX900 with TDP 3 Watt.

The total power consumption for this evaluation platform is approximately 10 Watts. The theoretical processor performance of VIA Eden X2 U4200 is 8.92 GFLOPS.

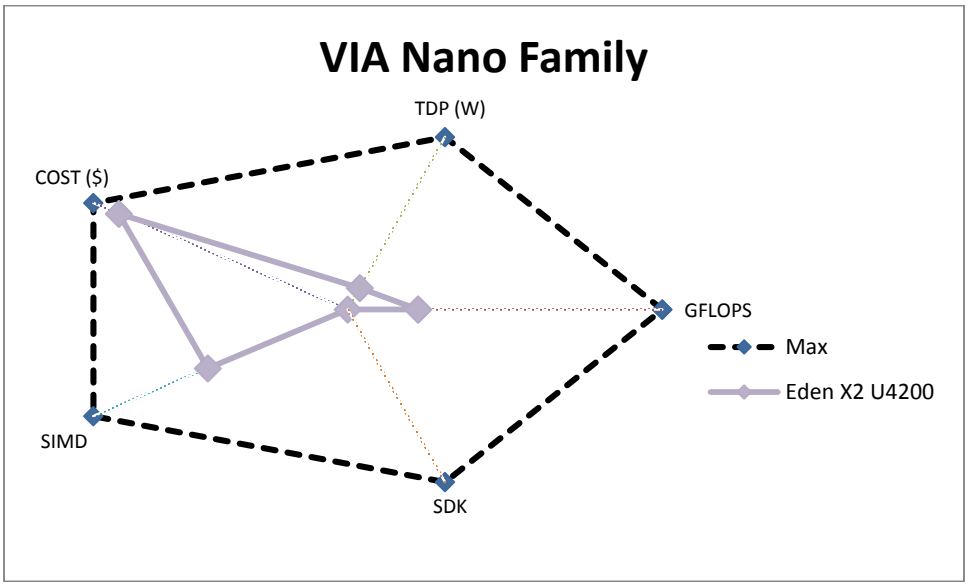


Figure 12. VIA Nano family performance chart

2.2.1 Other x86 Families

There are also other x86 processor design manufacturers such as DM&P Electronics' Vortex86 family, but due to lack of time and the difficulty of locating a proper reference design, these minor players x86 families are not included in the evaluation process.

2.2.1 Conclusion and Decision of x86 CPU

When it comes to efficiency, VIA Nano platform provides the best GFLOPS per watt, however, it is also the most expensive and non-existent software API support which is due to the fact that VIA does not provide any SDK or API without NDA (None Disclosure Agreement).

AMD APU is a very interesting product and the future of processors is heterogeneous architecture combining CPU and GPU. However, AMD's implementation is still at its infancy and the software API

supports and documentations are very lacking and at the very least: chaotic with disarray. The power efficiency is quite low and AMD has no dedicated developer support team.

Intel, on the other hand, has a processor covering the entire spectrum with very competitive pricing. Intel's 1st generation ATOM really lacks in both performance and power efficiency. But the efficiency as corrected by integrating Northbridge into the processor is still lackluster.

Intel i-Core does provide the best of all worlds, but the pricing is quite high due to the newness in the market. That leads to the Core 2 family. Core 2 family in dual core does provide comparable performance to i-Core, but due to lack of many modern SIMDs it is unable to compete when i-Core's SIMD and API is fully utilized. So the brute force to combat that problem is to increase the core counts to "4", and as a result Intel Core 2 Quad mobile 9100 became the choice by default.

On a technical paper published by Intel, they indicated that the Core 2 Quad 9100 [25] is about 33% more efficient than i3-390M [27]/i5-560M [28]/i7-620M [29], however, in the modern benchmark and more optimized application, PassMark [30] has shown that Core 2 Quad 9100 scores 3217, i3-390M scores 2321, i5-560M scores 2621, and i7-620M scores 2723. The performance difference varies from 33% to 15%, however, the power consumption is greatly reduced by up to 40% when utilizing the i-Core platform.

Clearly i-Core is a much better choice for this robot tank application, but due to limited funding and the likelihood of mishandling and damaging in the test phase, Intel Core 2 Quad mobile has been selected as the test platform.

Table 2. Summary of x86 CPUs

Name	GFLOPS	TDP (W)	COST ($)	SIMD	SDK	Motherboard	SDK Type	SIMD Type
ATOM 330	6.4	35	180	8	2	ASUS AT3IONT-I DX	Thread, ATOM	MMX, SSE, SSE2, SSE3, SSSE3,EMT64, XD bit, HTT
ATOM D525	7.2	16	180	8	2	ASUS AT5IONT-I	Thread, ATOM	MMX, SSE, SSE2, SSE3, SSSE3,EMT64, XD bit, HTT
T9550	21.328	52	100	13	1	BCM MX45GM2	TBB[a]	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, EIST, EMT64, XD bit, iAMT2, Intel VT-x, TXT, IDA
Q9100	36.25	62	150	13	1	BCM MX45GM2	TBB	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, EIST, EMT64, XD bit, iAMT2, Intel VT-x, TXT, IDA
i3-390M	21.328	39	150	13	1	Advantech AIMB-270	TBB, iGP	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EIST, EMT64, XD bit, VT-x, HTT, Smart Cache
i5-560M	21.328	39	180	14	1	Advantech AIMB-270	TBB, iGP	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EIST, EMT64, XD bit, VT-x, HTT, Turbo Boost, Smart Cache
i7-620M	21.328	39	200	17	1	Advantech AIMB-270	TBB, iGP	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EIST, EMT64, XD bit, TXT, VT-x, VT-d, HTT, Turbo Boost, AES-NI,Smart Cache
A8-3550MX	27	50	250	10	1	AMD reference Sample	APU	SSE, SSE2, SSE3, SSE4a, Enhanced 3DNow!, NX bit, AMD64, PowerNow!, AMD-V, Turbo Core
E350	7.4	24	120	9	1	ASUS E35M1-I	APU	SSE, SSE2, SSE3, SSSE3, SSE4a, NX bit, AMD64, PowerNow!, AMD-V
Eden X2 U4200	8.92	10	270	11	0	VIA EPIA-M900	NONE	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, x86-64, NX bit, x86 virtualization, VIA PadLock (SHA, AES, RNG), VIA PowerSaver

[a] TBB = Intel Threading Building Block

2.3 General Purpose Graphic Processor

There are 3 major players in Graphic Processor Unit (GPU), Intel, AMD/ATI, and NVidia present. All of the GPU solutions from each company not only accelerates 3D, rendering, and gaming but also is rich in computational functions through proprietary API.

Back in the 1990's, the majority of the video cards on the market were only capable of 2D rendering and in order to render 3D or polygons a "3D Accelerator" is required, which needs the assistance of CPU to off-load some of the rendering to the accelerator card. In 1995, with the rise of DVDs and digital video playback, many of the video chip design manufactures added video decoding capability to the ASIC design which allows MPEG decoding, video overlay and video post-processing to be offloaded to the graphic card.

In 2006, Nivida introduced GeForce 8 family, a new generation of generic steam processing unit which has the capability of parallel computation through the proprietary API: Compute Unified Device Architecture (CUDA [33]). The era of the General Purpose Graphic Processor Unit (GPGPU) has begun.

In 2009, rival AMD/ATI introduced their own version of GPGPU, named the Radeon HD 5000 family, and in 2010 Intel developed the Sandy Bridge's 2nd generation Intel HD Graphics to support GPPGU for parallel computation. Both AMD/ATI and Intel utilize the open source GPGPU computational language: OpenCL.

To properly evaluate the GPGPUs, the power TDP will be limited to less than 60 Watts TDP. 30 Watt TDP can be powered natively though PCI-Express 1x bus and anything less than 75 Watt TDP can also be natively power through PCI-Express 16x bus without additional +12 volt rail.

2.3.1 NVidia GPU

NVidia is considered as the pioneer in introducing GPGPU for the consumer mass market. The first generation of NVidia GPGPU is the G80 architecture, followed by the G90 which is a very minor die shrink with very little features updated for the GPGPU support.

2.3.1.1 NVidia GT200 Family

It is a NVidia's 11th generation GPU with the 2nd generation unified shader's architecture. It is a first major update over the original unified shader's architecture which is used in both GeForce 8 (G80) and GeForce 9 (G90) NVidia GPU.

The majority of the updates in GT200 (CUDA 1.3) are addressing the limitation and short coming of the 1st generation unified shader's architecture, such as the maximum number of warps, threads, atomic functions operation and share memory optimization.

The GT200 became the primary GPGPU card for scientific computation, research and other applications that require high performance computation.

However, any GeForce less than the 260 are not GT200 architecture, but are instead merely rebranded versions of the GeForce 9 in GeForce 200 family. GeForce 260 features 192 shader's but with more than 200 Watt TDP, and as a result, this card is not an ideal choice for the robot tank.

2.3.1.2 NVidia Fermi Family

NVidia Fermi is a NVidia 11th generation GPU with the 3rd generation unified shader's architecture, which is another major update of the NVidia GPGPU with a focus on the general purpose computation for workloads encompassing tens of thousands of threads.

NVidia Fermi family spread across 2 series: GeForce 400 and GeForce 500 series. The GeForce 500 is a significantly modified version of 400 series in terms of performance and power management. In return, the GeForce 500 offers much more power savings and higher efficiency.

GeForce GT 430 "GF108 - Fermi 1.0" has 96 cores, 268.8 GFLOPS, and TDP at 50 Watts and GeForce GT 440 "GF106 - Fermi 1.0" has 144 cores, 342.43 GFLOPS, and TDP at 56 Watts. GeForce GT 520 "GT119 - Fermi 1.1" has 48 cores, 155.5 GFLOPS, and TDP at 29 Watts, and GeForce GT 530 "GT118 - Fermi 1.1" has 96 cores, 268.8 GFLOPS, and TDP at 50 Watts.

2.3.1.3 NVidia Kepler Family

NVidia Kepler architecture is the most current and highest performance NVidia GPU architecture up-to-date. This architecture addresses the efficiency, throughput, and more developer oriented hardware features [34].

Dynamic Parallelism: It adds the capability for the GPU to generate new work for itself, synchronize on results, and control the scheduling of that work via dedicated, accelerated hardware paths, all without involving the CPU. By providing the flexibility to adapt to the amount and form of parallelism through the course of a program's execution, programmers can expose more varied kinds of parallel work and make the most efficient use of the GPU as computation evolves. This capability allows less structured and more complex tasks to run easily and effectively, enabling larger portions of an application to run entirely on the GPU. In addition, programs are easier to create, and the CPU is freed for other tasks.

This is especially useful in CUDA programming because you can dynamically create any objects within the CUDA program, instead of return back to CPU code and recreating the wanted object, as shown in Figure 13.

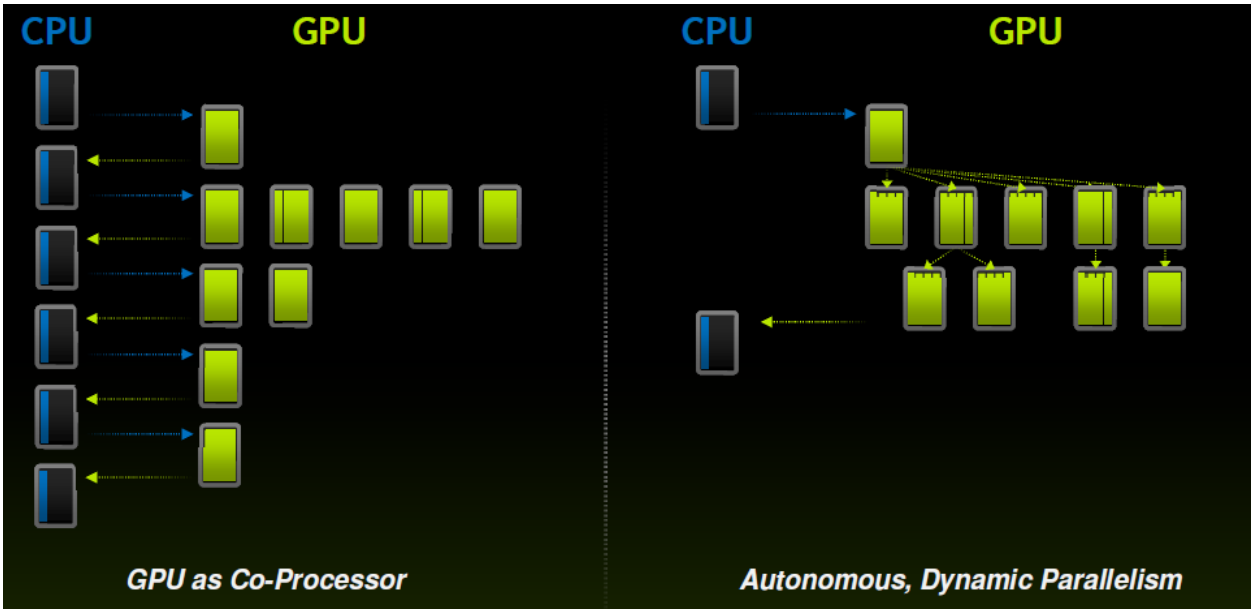


Figure 13. Dynamic parallelism for NVidia Kepler architecture

Hyper-Q: Hyper-Q enables multiple CPU cores to launch work on a single GPU simultaneously, thereby dramatically increasing GPU utilization and significantly reducing CPU idle times. Hyper-Q increases the total number of connections (work queues) between the host and the GK110 GPU by allowing 32 simultaneous, hardware-managed connections (compared to the single connection available with Fermi). Hyper-Q is a flexible solution that allows separate connections from multiple CUDA streams, from multiple Message Passing Interface (MPI) processes, or even from multiple threads within a process. In other word, for an application that previously encountered false serialization across tasks, hence, limiting GPU's full utilization, the Hyper-Q can dramatically improve performance without changing any existing code.

Due to the different grid configuration, the utilization of CUDA cores in Fermi may not be optimum. Kepler architecture allows multiple core configuration and utilization happening at same time. This is especially important when it comes to squeezing the full potential of the CUDA computation capability, see Figure 14.

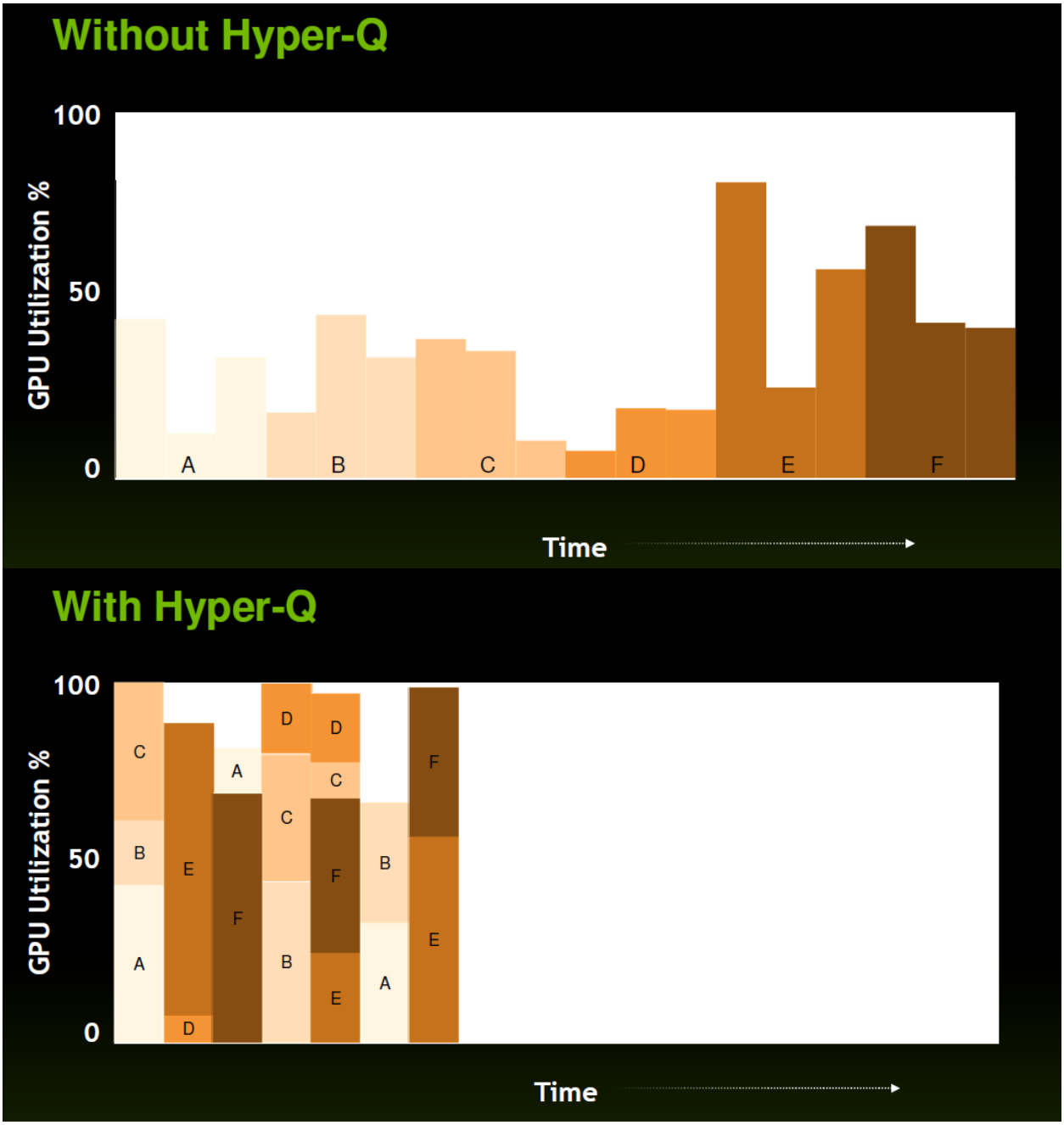


Figure 14. GPU utilization with Hyper-Q

Grid Management Unit: Enabling dynamic parallelism requires an advanced, flexible grid management and dispatch control system. The new GK110 Grid Management Unit (GMU) manages and prioritizes grids to be executed on the GPU. The GMU can pause the dispatch of new grids and queue pending and suspended grids until they are ready to execute, providing the flexibility to enable powerful runtimes, such as dynamic parallelism. The GMU ensures both CPU- and GPU-generated workloads are properly managed and dispatched.

In NVidia Fermi architecture, there is a maximum amount of 16 GPU grid blocks with each grid composed of lots of threads blocks, where threads are arranged into a shape of block in 2D or 3D, as shown in Figure 15. This becomes problematic for the developer due to the limited amount of grids configuration. In many cases, the high end CUDA card's capability has been limited to the amount of grids that can be present in the program.

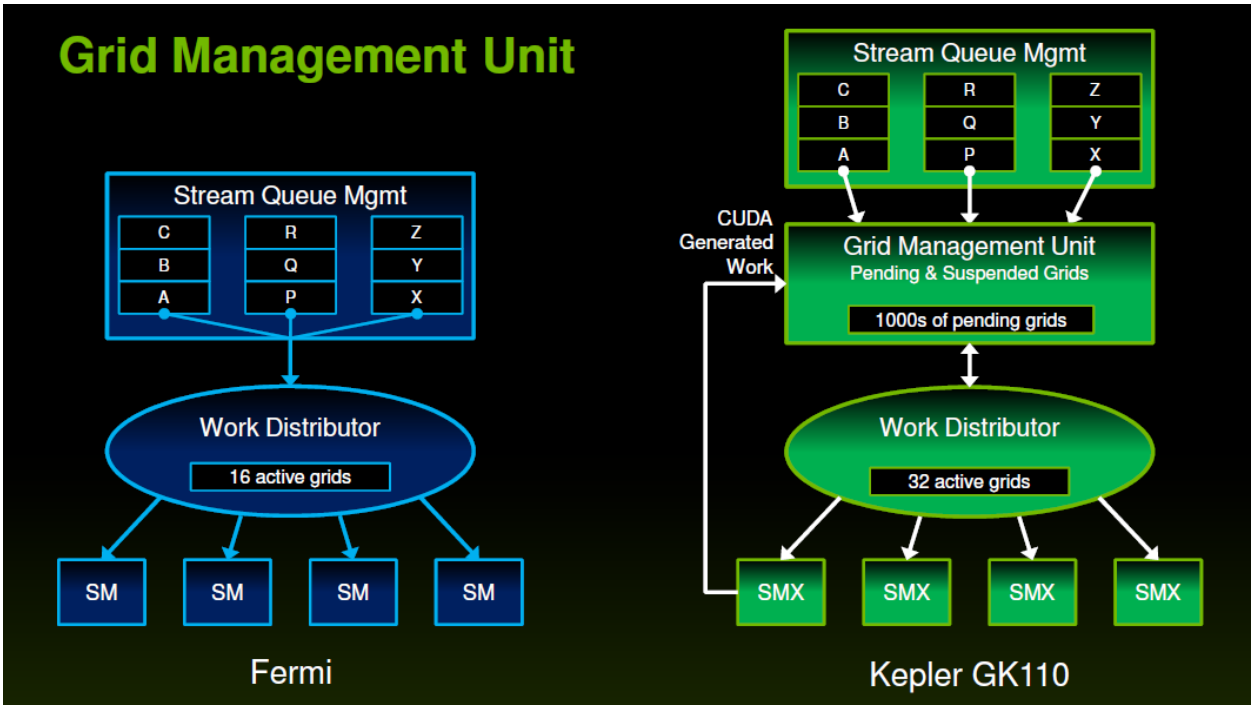


Figure 15. Grid management for NVidia Kepler architecture

NVidia GPU Direct™: NVIDIA GPUDirect™ is a capability that enables GPUs within a single computer, or GPUs in different servers located across a network, to directly exchange data without needing to go to the CPU/system memory. The RDMA feature in GPU Direct allows third party devices such as Solid State Drives (SSDs), Network Interface Cards (NICs), and Infinity Band (IB) Host adapters to directly access memory on multiple GPUs within the same system, significantly decreasing the latency of MPI send and receive messages to/from GPU memory. It also reduces demands on the system memory bandwidth and frees the GPU DMA engines for use by other CUDA tasks. Kepler GK110 also supports other GPUDirect features including Peer-to-Peer and GPU Direct for Video.

2.3.1.4 Conclusion and Decision of NVidia GPU

Every generation of NVidia's GPU architecture brought additional functionality and increased the limit of the previous generation as shown in **Table 3.** NVidia's Kepler architecture brought hyper-Q and dynamic parallelism which are both extremely useful for image processing. Dynamic parallelism allows me to dynamically scale up and down the resolution/accuracy in Region of Interest (ROI) and with hyper-Q, I can predefine different grid sizes for each different image filters.

Table 3. NVidia GPU capability chart

	G8x/G9x/G200	Fermi GF100	Fermi GF110	Kepler GK104	Kepler GK110
HW Capability	1.0-1.1/1.2-1.3	2.0	2.1	3.0	3.5
Threads/Warp	32	32	32	32	32
Max Warp	24/32	48	48	64	64
Max Threads	768/1024	1536	1536	2048	2048
Max Thread Blocks	8	8	8	16	16
32-bits Registers	24576/32768	32768	32768	65536	65536
Max Registers	128	63	63	63	255
Max Threads	512	1024	1024	1024	1024
Share Memory Size Config	16K	16K/48K	16K/48K	16K/32K/48K	16K/32K/48K
Max 1D Grids	65535	2^{16}-1	2^{16}-1	2^{32}-1	2^{32}-1
Hyper-Q	NO	NO	NO	NO	YES
Dynamic Parallelism	NO	NO	NO	NO	YES

However, NVidia decided to create 2 family lines with Kepler architecture: GK104 and GK110. Technically both are identical down to the silicon level, but the advance features have been disabled on the main stream GPU - GK104 and made only available on the high GPU – GK110.

The GK110 is a high-end product. The highest performance is available with the sacrifice of high energy consumption and proportionally high thermal output. This is not ideal for a mobile robot. As a result, Kepler family is no longer an option.

NVidia Fermi is a mature product. GF110 is a significantly modified version of GF100 in terms of performance and power management. This architecture did not artificially cap the CUDA capability in order to diversify the market. GK104 limited 64-bits double-precision math performance to 1/24 of single-precision, while both Fermi GF100 and GF110 is only ½ of single-precision.

Based on the best watt per flops, GeForce 520 and GeForce 530 is the most suitable choice for mobile application.

2.3.2 ATI/AMD VPU

ATI/AMD has competitive VPU for GPGPU purpose. However, due to the lack of delicate development team for GPGU, obstacles, such as immaturity of the OpenCL development tools and drivers, prevent me to fully evaluate ATI/AMD VPU for this mobile robot tank GPU computation assistance.

2.4 Mouse Pointers

All of the vehicles utilize tachometer to keep track of traveled distance by measuring the rotational speed of a shaft or disk. However, this technique has one major flaw: the tracked distance does not

translate to the actual travel distance, as the tire or treads on the vehicle may have slipped during the travel and this will translate to an error in the recorded travel distance.

With that in mind, I propose an off-the-shelf solution by utilizing mouse pointer for the traveled distant sensor. Mouse pointer's movement on the screen can be mathematically correlated to actual travel distance, as the movement in pixels and actual travel is one to one ratio. However, some of the modern mouse pointer incorporates travel acceleration, which makes the pixel to travel no longer one to one ratio.

All the modern mouse pointers are utilizing one or more optical technologies: laser or LED. Mouse pointers are designed to lay flush and parallel to the surface of travel. The problem of possible collision will occur when attempting to mount the mouse pointer to the bottom of the mobile robot. To minimize any potential travel or ricochet damage due to traveling, the mouse pointer should be at a height or at least some minimum distance away from the parallel surface.

Different mouse pointers have different sensitivities to the height. As a result, I acquired few computer mouse pointers for evaluation and experiment to determine the most suitable mouse pointer for this mobile robot tank based on the following criteria: cost of ownership, maximum z-axis, communication protocol, and ease of modification.

2.4.1 Selections and Test Results of Mouse Pointer

The following mouse pointers are evaluated and tested:

- Microsoft Wheel Mouse Optical
- Microsoft IntelliMouse
- Microsoft IntelliMouse Explorer
- Microsoft SideWinder X8
- Microsoft Wireless Laser Mouse 6000
- Logitech G9x
- Other No-Brand Optical Mouse

After testing LED optical sensor mice, the color and brightness of the sensor has direct influences to the z-axis and different surfaces affect the accuracy of the read out, as shown in **Table 4**.

Table 4. LED color and depth

Color	Default Voltage (2V)	Low Voltage (3V)	High Voltage (3.5V)
Red (stock)	3.6 mm	3.6 mm	3.8 mm
White	0 mm	1.8 mm	4.9 mm
Blue	0 mm	1.8 mm	5.0 mm
Green	1 mm	1 mm	1 mm
Yellow	0 mm	0 mm	0 mm

The red LED is the most commonly used LED for optical mouse. The test indicated that under the default voltage it outperforms all the other colors. However, when the voltage increases, the brightness increases, but the color spectrum decreases. This results in minimum gain in height.

The white and blue colors LEDs show no sign of light when connected to default voltage because both blue and white LEDs require a minimum of 3V forward voltage. At 3V it performs worse than red LED, but as soon as high voltage applies, the brightness drastically increases and has proportional effect to the gain in height. The test shows that white LED and blue LED are at par with each other during the test. However, if the surface is much less reflective, such as a dark surface, the blue LED sustains the similar height accuracy, but the white LED's height has to reduce to prevent cursor jumping.

The other color such as green and yellow does not provide any advantage over red. In fact, green LED performs much worse than red LED in all three voltages. On the other hand yellow LED cannot be registered by the mouse sensor in all 3 voltages.

The red LED is the most cost-effective, while the white and blue LEDs are the most expensive. This explains why most manufactures prefer red LED over all the other colors. In 2007, Microsoft introduced "BlueTrack Technology" for high-end and luxury models of the Microsoft mouse. Now all of the Microsoft optical mice have been updated with "BlueTrack Technology".

The Microsoft BlueTrack Technology is essentially a blue LED instead of a regular red LED. The test demonstrated that blue LED has a competitive advantage over other colors: higher accuracy on different surfaces, high depth tracking and low power consumptions in comparison to laser diode.

The other type of optical mouse utilizes the laser diode. Microsoft Wireless Laser Mouse 6000 and Logitech G9x are the only two mice that have laser optical sensor. The laser diode has much higher height tracking, especially with the Logitech G9x. The Logitech G9x can achieve up to 10 mm with white surface, however, half the height with reflective surface and unusable on a surface such as glass.

The Microsoft wireless laser mouse 6000 has only quarter the height of the Logitech counterpart, which is due to the attempt to conserve battery power for maximum usage, and as a result, the voltage for the laser is also lower than the wired mouse.

2.4.2 Conclusion and Decision of Mouse Pointer

The only suitable choice for the laser mouse pointer is the Logitech G9x incredible height tracking. For the LED mouse pointer, any of the mouse pointers are acceptable with the modification of the white LED.

2.5 Sensor Selection Conclusion

Microsoft Kinect sensor has multiple roles in the mobile robot tank build. It is utilized as distance sensor, imaging sensor, and acoustic sound sensor for an all-in-one sensor solution. Intel Core 2 Quad 9100 is paired up with NVidia Fermi family's 520/430 GPU for the computational purpose due to the best energy consumption to performance ratio. The Logitech G9x mouse pointer is chosen for travel tracking due to the incredible height tracking capability.

Chapter 3 System Implementation

The robot tank is a tread based mobile robot, with multiple types of sensors to and from an environment self-awareness system. This system consists of 8 major parts: Digital Signal Processor (DSP), Single Board Computer (SBC), Graphic Processing Unit (GPU), Kinect sensor, proximity sensors, Power Distribution System (PDS), pointer tracking feedback, and Motor Control System (MCS), as shown in Figure 16.

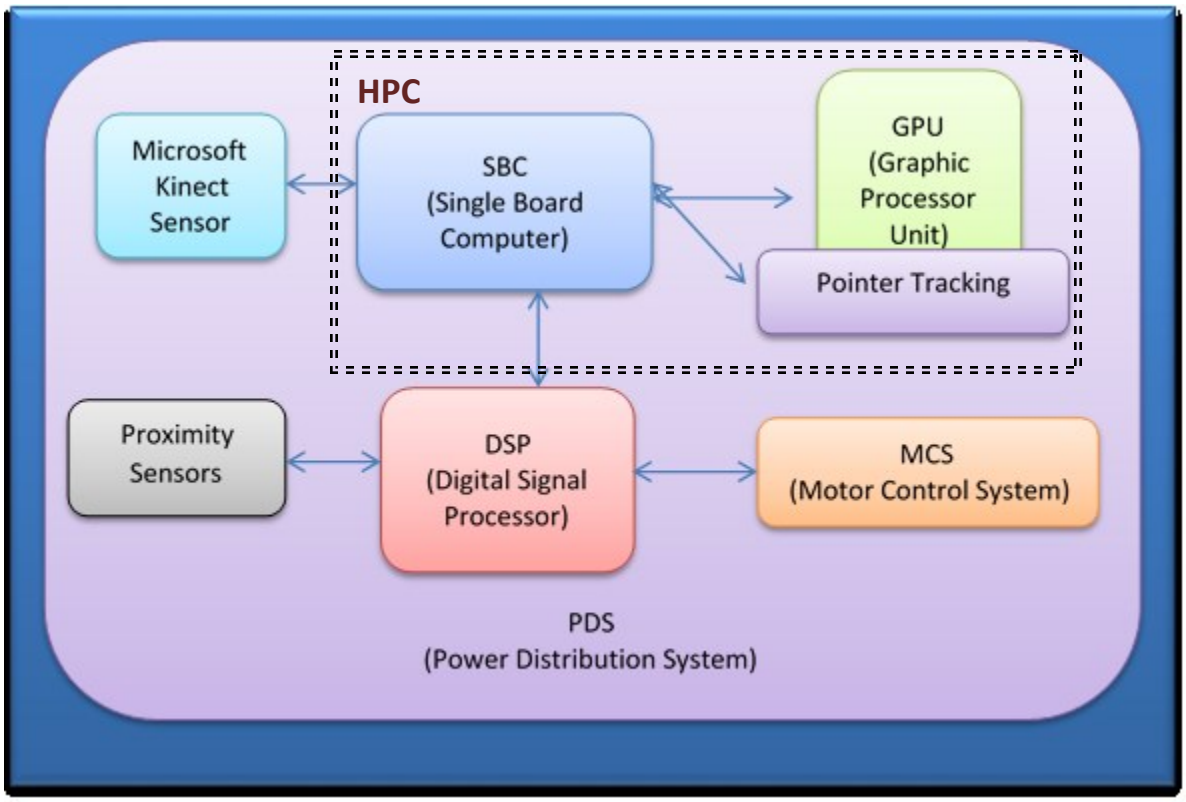


Figure 16. Block diagram for the mobile robot tank control system

The SBC acts like a command center of the robot and provides decisions based on data from the Microsoft Kinect sensor. However, due to the sheer amount of raw data that is delivered and the analysis and computation that is required, this bulk load of data is shared by both the CPU and the GPU.

Once properly analyzed and computed, the motor control directive will be issued to the DSP, which will allow the DSP to pass the control information to the MSC (Motor Control System). At the same time, the proximity sensors also provide additional information regarding the current location of the robot and "Just-In-Time" feedback to prevent damage to the robot itself.

Any movement or relocation will be tracked by the onboard mouse pointer for both recording and traveling distance feedback. The recorded information can be reused for step retracing and aids of environment awareness in the future work.

3.1 Utilized SDK, API, and IDE

To achieve rapid prototyping without designing everything completely from the ground up, I utilize some first party Kinect SDK from Microsoft for Kinect raw data retrieval, Microsoft speech platform SDK for language development and voice recognition engine, OpenCV for all the image processing filters and algorithms, and Emgu as C# wrapper to interpolate C++ to C#. NVidia CUDA is primarily used to increase the image processing performance. CUDA is a C base language requiring a wrapper to interpolate to C# by utilizing managed CUDA. Mouse pointer tracking is built on SlimDX API for accessing DirectX API in .Net. To ensure the portability and ease of management, the entire software is built on top of Microsoft .Net framework with Compact .Net Framework for mobile devices support.

All of the development is done on Microsoft Visual Studio 2010 with Async CTP 3.0 and Direct X 9 in Microsoft Windows 7 environment.

3.1.1 Microsoft Kinect SDK 1.5 – 1.7

Microsoft Kinect SDK is the first party software and driver support for Microsoft Windows. There are other SDK and software packages by the open source community, called OpenNUI. However, maturity and stability of these third party packages are not guaranteed at the time of release and testing.

Microsoft Kinect SDK for windows provides a sophisticated library to aid developers to interact with Kinect-based natural input, as shown in Figure 17.

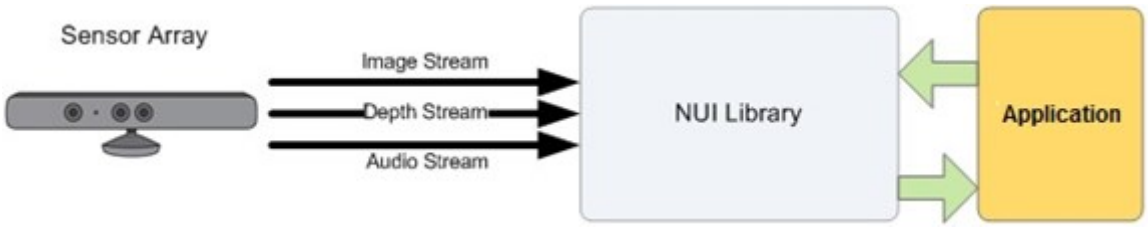


Figure 17. Kinect hardware and software interaction with developer's applications[3]

To simplify the development process and remove any learning curve, Microsoft Kinect SDK for Windows removes the needs of know and how in Kinect hardware interfacing and software stack, which is done by a simple function call to access all the desired information.

[3] Microsoft MDSN Kinect for Windows SDK – Programing Guide (http://msdn.microsoft.com/ens/library/jj131023.aspx)

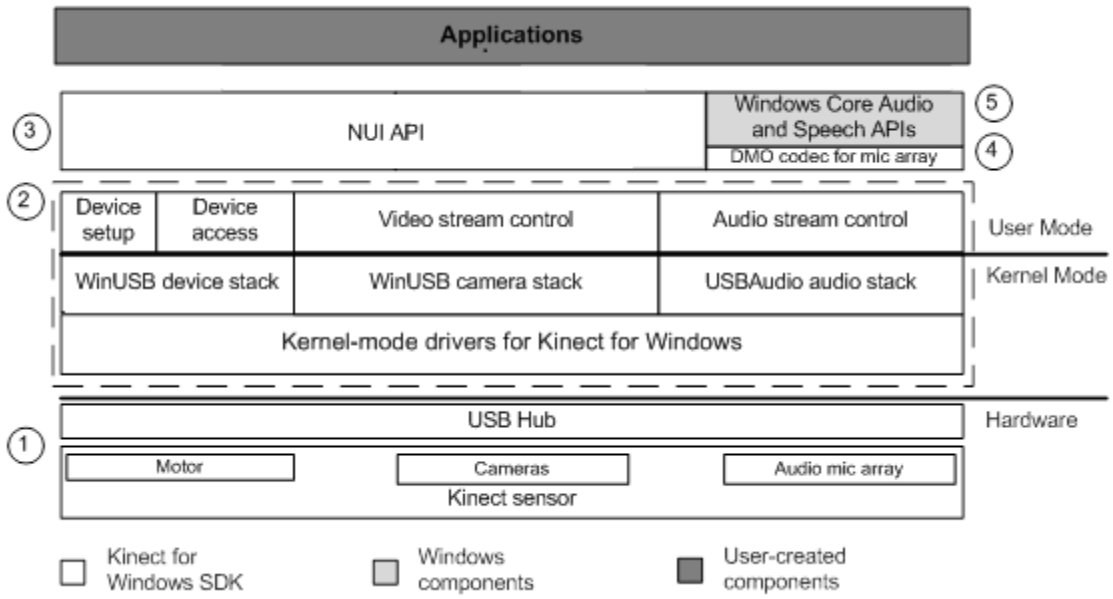


Figure 18. Kinect SDK for Windows architecture[4]

Kinect SDK for Windows architecture [35] includes the following components, as shown in Figure 18.

1) Kinect hardware is mainly composed of sensors and a USB hub through which the sensors are connected to the computer.
2) Kinect drivers are computer programs that operate or control Kinect, which are installed as part of the SDK setup process. The Kinect drivers provide support for the Kinect microphone array, audio and video streaming controls for streaming audio and video, and device enumeration functions that enable an application to use more than one Kinect.
3) Audio and video components are Kinect natural user interface for skeleton tracking, audio signals, and color and depth imaging.
4) DirectX Media Object (DMO) is designed for microphone array beam forming and audio source localization.
5) Windows 7 standard APIs are the audio, speech, and media APIs in Windows 7. These APIs are also available to desktop applications in Windows 8.

The Kinect SDK for Windows provide reusable code and functions in both C++ and C#. However, some of the prebuilt algorithms and filters can only be used in C# with Windows Presentation Framework (WPF).

3.1.2 Microsoft Speech Platform and SDK

Microsoft speech platform and SDK utilize the pre-installed speech recognition engine that is bundled with Microsoft Window Vista or newer.

[4] Microsoft MDSN Kinect for Windows SDK – Programing Guide (http://msdn.microsoft.com/en-us/library/jj131023.aspx)

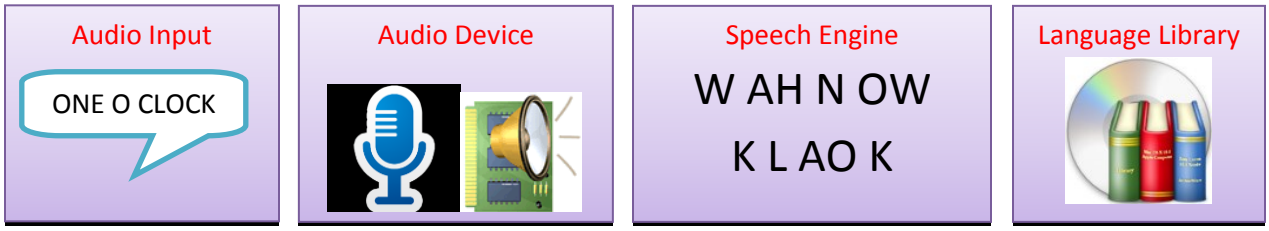


Figure 19. Speech recognition process

For any speech recognition, there are 4 major components, as shown in Figure 19.

1) Audio input is a voice command or phrase issued by the user.
2) Audio device is the device in which the analog audio wave converts into a digitization format.
3) Speech engine is the analysis process of input audio wave and determination of possible phrase.
4) Language library is a database composed of digital information for words and phrases. Once the possible phrase is determined, the speech engine will go into the language library to determine the matching words or phrase.

Microsoft speech platform eases the developing process by unifying the entire process. The developer no longer needs to worry about the diverse manufacturers of sound card or audio device, nor do they need to worry about issues related to the sample rate, sensitivity, driver and compatibility. Microsoft address the audio device issues by making sure that all the sound cards are Direct X compliant with the predefined software and driver guideline.

There are two basic technologies for computers to process speech signals: Speech Recognition (SR) and speech synthesis, depending on who is doing the talking, user or computer. Microsoft speech platform incorporated both speech recognition engine (speech to text) and speech synthesizer (text to speech) capabilities. [36]

3.1.2.1 Speech Synthesizer

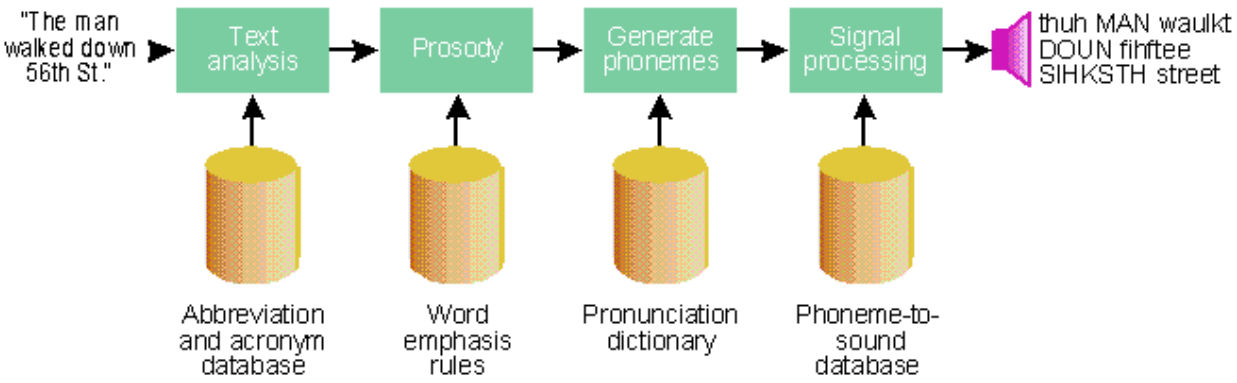


Figure 20. Text-to-speech engine

Speech synthesis is commonly called Text-To-Speech (TTS) since the speech is usually synthesized from text data. Figure 20 shows the architecture of a typical text-to-speech engine.

The process begins when the application sends a string of text to the engine, such as, "The man walked down 56th St." The text analysis module converts numbers into words, identifies punctuation such as commas, periods, and semicolons, converts abbreviations to words, and even figures out how to pronounce acronyms. Some acronyms are spelled out (example of acronym: MSJ) whereas others are pronounced as a word (example of acronyms word: FEMA). The sample sentence would get converted to something, as shown in Figure 21.

```
<beginStatement>
The man walked down fifty sixth street
<endStatement>
```

Figure 21. Sample XML text-to-speech snippet

Text analysis is quite complex because written language can be very ambiguous. A human being has no trouble pronouncing "St. John St." as "Saint John Street," but a computer, in typically mechanical fashion, might come up with "Street John Street" unless a clever programmer gives it some help.

Once the text is converted to words, the engine figures out what words should be emphasized by making them louder or longer, or giving them a higher pitch. Other words, on the other hand, may be deemphasized. Without word emphasis, the result is a monotone voice that sounds robotic, like something out of a 1950s sci-fi flick. After adding prosody, the sample sentence might end up with something as shown in Figure 22.

```
<beginStatement>
<de-emphasize>the <emphasize>man walked
<emphasize>down fifty <emphasize>sixth street<pause>.
<endStatement>
```

Figure 22. Resultant sample sentence snippet

Then, the text-to-speech engine determines how the words are pronounced, either by looking them up in a pronunciation dictionary or by running an algorithm that guesses the pronunciation. Some text strings have ambiguous pronunciations, such as "read." The engine must use context to disambiguate the pronunciations. The result of this analysis is the original sentence expressed as phonemes. "Th-uh M-A-Nw-au-l-k-tD-OU-Nf-ih-f-t-eeS-IH-K-S-TH s-t-r-ee-t".

Finally, the phonemes are parsed and their pronunciations are retrieved from a phoneme-to-sound database that numerically describes what the individual phonemes sound like. If speech were simple, this database would have only forty-four entries, one for each of the forty-four English phonemes (or whatever language is used). In practice, each phoneme is modified slightly by its neighbors, so the database often has as many as 1600 or more entries. Depending on the implementation, the database might store either a short wave recording or parameters that describe the mouth and tongue shape.

Either way the sound values from the database are smoothed together using signal processing techniques to form a digital audio signal, which is sent to an output device such as a PC sound card.

3.1.2.2 Speech Recognition

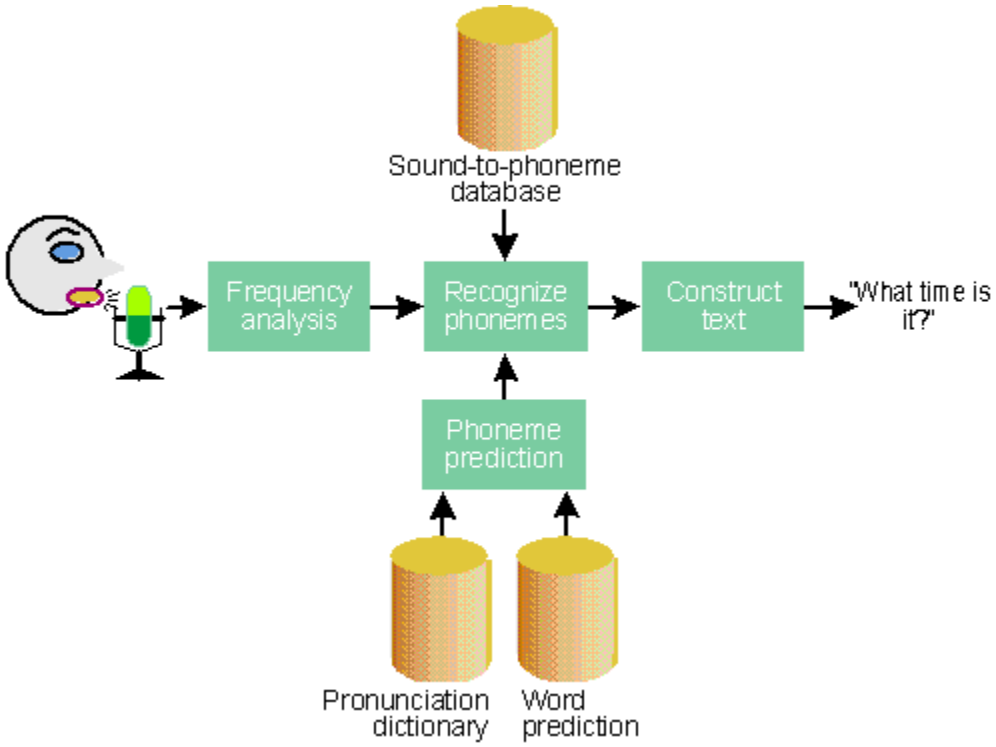


Figure 23. Speech recognition engine

Figure 23 shows a generic speech recognition engine. When the user speaks, the sound waves are converted into digital audio signals by the computer's sound card. Typically, the audio signals are sampled at 11 KHz with 16 bits. The raw audio signals are first converted by the frequency analysis module to a more useful format. This involves a lot of digital signal processing that is too complicated to describe here. The basic challenge is to extract the meaningful sound information from the raw audio data. If you were to say the word "foo," and then say "foo" again, then the waveforms generated would look kind of similar. But there is no way you could consistently recognize them as the same sound by a simple comparison, without applying some pretty hairy mathematical techniques using Fourier transforms transformations. Fortunately, people have already figured this stuff out.

The converted audio data is then broken into phonemes by a phoneme recognition module. This module searches a sound-to-phoneme database for the phoneme that most closely matches the sound it heard. Each database entry contains a template that describes what a particular phoneme sounds like. As with text-to-speech, the database typically has several thousand entries. While the phoneme database could in theory be the same as that used for TTS, in practice they are different because the SR and TTS engines usually come from different vendors.

Because comparing the audio data against several thousand phonemes takes a long time, the speech recognition engine contains a phoneme prediction module that reduces the number of candidates by predicting which phonemes are likely to occur in a particular context. For example, some phonemes rarely occur at the beginning of a word, such as the "ft" sound at the end of the word "raft." Other phonemes never occur in pairs. In English, an "f" sound never occurs before an "s" sound. But even with these optimizations, speech recognition still takes too long.

A word prediction database is used to further reduce the phoneme candidate list by eliminating phonemes that don't produce valid words. After hearing, "y eh," the recognizer will look for "s" and "n" since "yes" and "yen" are valid words. It will also search for "m" in case you say "Yemen." It will not interpret it for "k" since "yek" is not a valid word. The candidate list can be reduced even further if the application stipulates that it only expects certain words. If the application only wants to know if the user said "yes" or "no," the phoneme recognizer needn't listen for "n" following "y eh," even though "yen" is a word. This final stage reduces computation immensely and makes speech recognition feasible on a 33MHz 486 or equivalent PC. Once the phonemes are recognized, they are parsed into words, converted to text strings and passed to the application.

3.1.3 OpenCV / Emgu

Open source Computer Vision library (OpenCV) is an open source library consisting of programming functions mainly focused on real-time computer vision, initially developed by Intel and now it is free for use under open source BSD (Berkeley Software Distribution) license.

OpenCV's main utilization is as follows:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–Computer Interaction (HCI)
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Stereo vision: depth perception from 2 cameras
- Structure From Motion (SFM)
- Motion tracking
- Augmented reality

OpenCV also features statistical machine learning library that contains:

- Boosting (meta-algorithm)
- Decision tree learning
- Gradient boosting trees

- Expectation-maximization algorithm
- K-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support Vector Machine (SVM)

OpenCV is written in C++ and retains C interface compatibility. However, there are wrappers in other languages such as C#, VB#, Ch… etc. In the mobile robot tank, we utilize Emgu, an OpenCV C# wrapper.

Emgu allows the developer to utilize the simple nature of C# without the need to invoke through C++. With Emgu, all the OpenCV function structures that are retained are almost identical. This allows code translation and migration much easier.

Emgu has been utilized fully for the majority of CPU image processing functions in the mobile robot tank controller and some of the functions are being ported to NVidia CUDA for GPU image processing.

3.1.4 NVidia CUDA / managedCUDA

A proprietary C++ API is for NVidia's GPU only. This API provides 2 levels of software support: runtime API and driver API. The CUDA runtime API provides portability for CUDA programming, as all the required Dynamic Linking Libraries (DLL) and Libraries (LIB) are deployed through the CUDA runtime library package during your application installation. Another advantage of the runtime API is that there is no need for CUDA version control and the runtime library takes care of it for you. On the contrast, the driver API allows direct manual control of the NVidia GPU hardware and resource management and also allows developers to determine which module is required for computation and to compile necessary DLLs for the runtime.

There are 2 ways to utilize NVidia CUDA for C#, one is using extern/invoke with CUDA C++ code or managedCUDA. ManagedCUDA is an open source CUDA C# wrapper for both Driver and API CUDA code. This wrapper has a ratio of 1:1 representation of C++ CUDA in C#, which, in other words, makes C# migration easier.

The mobile robot tank utilizes the NVidia Fermi architecture GPU and CUDA version 4 for the image processing functions.

3.1.5 SlimDX

Microsoft Direct X is dated all the way back to Windows 95, in hope of centralization and to create uniformity of experience across users with completely different hardware configurations. More than 10 years later, Direct X became the detector of hardware and driver standard, all the hardware has to be Direct X compatible and the feature sets are defined by the Direct X version. This approach simply helps the development process on accessing hardware level input devices and multimedia.

In order to access Natural User Interface (NUI), such as keyboard, mouse, gamepad … etc., developers are required to go through the Direct X layer in order to access those devices. However, the Direct X SDK

are currently only in C/C++, so, just like CUDA and OpenCV, a wrapper class is needed for C# implementation.

SlimDX is a .Net Framework wrapper for Direct X, which also retains the same syntax as the Direct X SDK's in C++. As a result the implementation is smooth and effective. This part of implementation is utilized for the mouse tracking in the mobile robot tank controller.

3.1.6 Microsoft .Net Framework and .Net Compact Framework

Microsoft .Net framework is a software framework for Microsoft Windows, developed by Microsoft in 2002. The design of .Net framework focuses on portability and transportability across different configurations of hardware, similar to the Java virtual machine environment.

The .Net framework comes with 2 variations per different revision, the full .Net framework and .Net compact framework. The .Net compact framework is designed for embedded and smart phone systems. The .Net framework eliminates the need of compiling the same software for different processor architectures and centralizes all the essential libraries into a standard development/deployment package.

Developing on .Net framework no longer needs to worry about missing DLLs, libraries, or APIs, as .Net framework centralizes and manages them in the background. This is a great advantage for future expansion of the mobile robot tank controller. Essentially a matured mobile robot tank will be deployed on a Windows CE system with embedded hardware.

3.3 Robot Tank Software Design and Implementation

There are two versions of mobile robot tank controller software: V1 and V2. The completion of the V1 mobile robot tank controller software gave me some insight and some different ideas of how to improve and properly design for the next version.

V1 is the first version that only acted as the proof of concept and only utilizes the depth information with almost no optimization at all. While V2 builds on top of the V1 design with additional color image processing, more threading, and GPU optimization. As a result, the V2 design consists of the same design concept as V1 but with a completely new design class and rewritten functions from the ground up to enhance the performance capability and scalability.

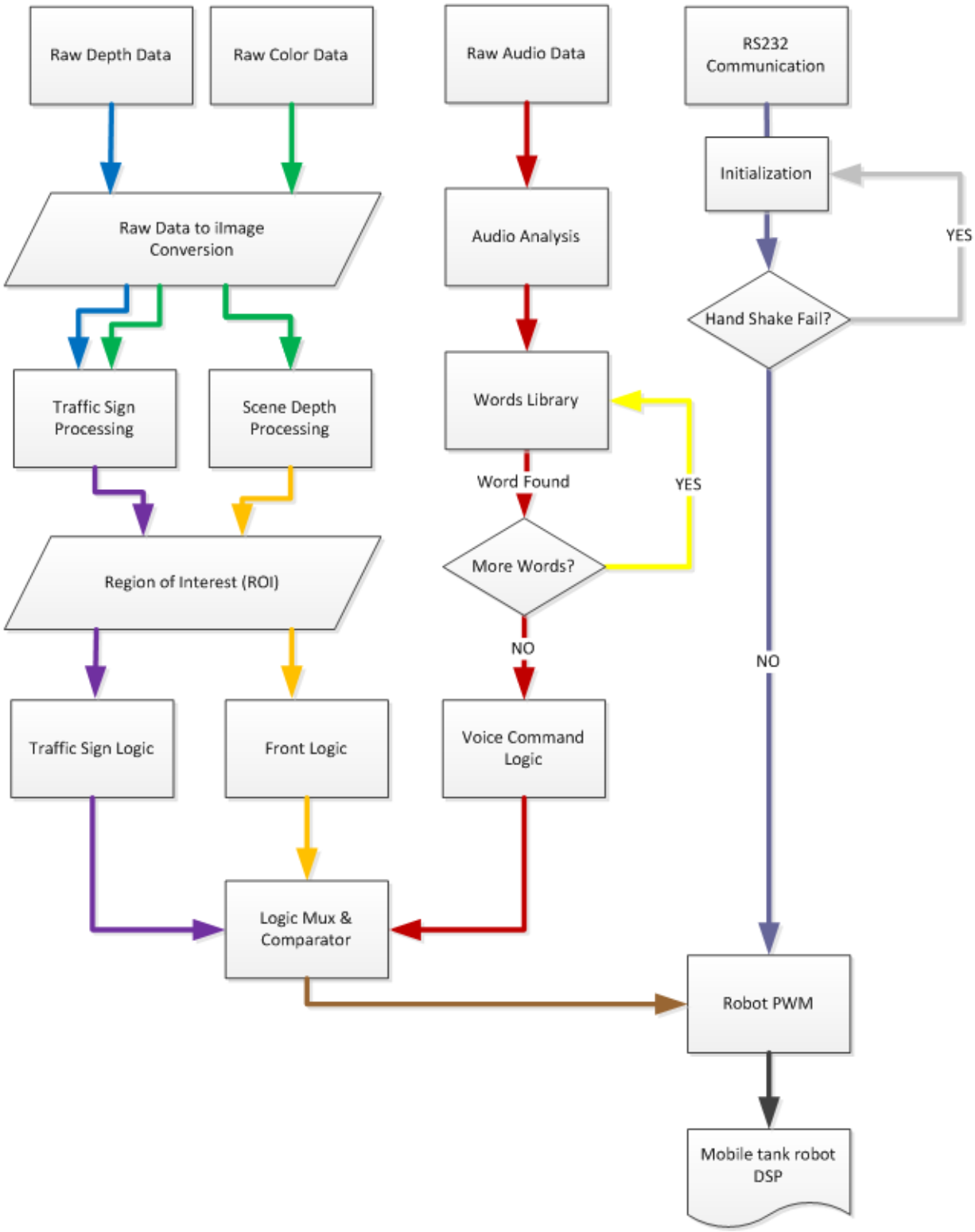


Figure 24. Mobile robot tank control software flow chat

Mobile robot tank controller software has 4 parts: raw data acquisition processing, audio signal processing, image signal processing and communication signal processing. The flowchart for the software design is shown in Figure 24.

The raw data acquisition process acquires a real-time depth data stream for image processing. Audio data acquisition only occurs when Kinect is being triggered by a sound signal and it will only record the voice until either the sound ceases or the maximum of 4 seconds passes, after which it resumes sleep mode. The raw data acquisition also acquires data from Kinect sensor's accelerometer based on the instance of request. The RGB color data is not the primary data source for mobile robot tank; however, when the request of RGB color data flag is set to "1", the raw data acquisition process will also stream the RGB color data until the flag set back to "0".

The audio process samples the raw audio clip for analysis, which consists of noise filtering, template matching for pronunciation, and voice command matching based on probability of possible outcomes and the highest probability outcome.

The image process in the mobile robot tank relies purely on the depth raw data. The process takes the raw depth data and divides it into sub-regions. Each sub-region has a threshold value corresponding to the real-time distance feedback. If the threshold is reached, that region will be flagged until the next frame arrives. Once all sub regions have been flagged, the mobile robot tank can determine the next logic command for travelling, based on the sub-region map.

Once the voice command and logic command have been processed, the corresponding result is then computed in the communication process for transmission with the DSP and SBC.

3.3.1 Raw Data Acquisition Processing

In Microsoft Kinect of Xbox 360, there are 3 major data streams and 1 data set. The 3 data streams consists of RGB color image, IR depth image, and audio stream from the microphone array. These 3 data streams can be accessed either by event trigger or pooling mode. Pooling mode requests the specific time stamped frame for processing and remains idle until the next request. Pooling mode has the advantage of minimum resource consumption, but may create problems when it comes to synchronization during multiple stream processing. The other mode is the event mode, which is designed for synchronization at desired frame rate. If "AllFramesReady" event is issued, it will synchronize both depth raw data and RGB color data at the lowest frame rate that is selected; if one of RGB or Depth raw data frame rates is higher, it will disregard the higher frame rate and synchronize with the lower value.

Mobile robot tank utilizes the event mode for RGB and IR depth data acquisition. The IR depth data is initialized at beginning. As soon as "DepthFramesReady" event is triggered, the program starts to stream IR depth data for processing at 640x480x30fps. This streaming process copies the data into computer RAM at 30 frames per second at the same allocation address, which means that there is only a very small window of opening to transfer the data to another location. To prevent a data accessing error, a lock is put on the same memory allocation during the copy time. If the copy time takes longer than expected, this may result in skipping the next frame.

3.3.1.1 V1 Design

The IR depth data is a short 1D array with a size of 307200 with each short data being a 16-bits raw data, as shown in Figure 25. The first 3 bits represent the identified players and remaining 13-bits are the distance in millimeters (mm). These 3 bits will be discarded for the mobile robot tank and the 13-bits distance data is processed with bitwise shifting operation to get the distance.

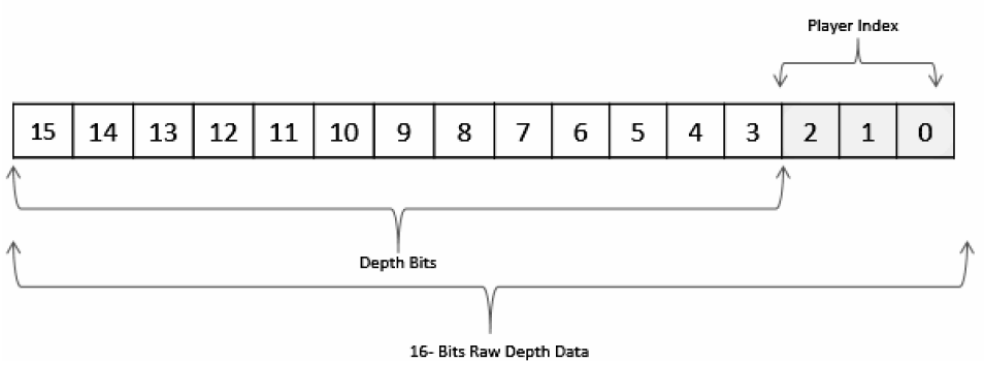


Figure 25. IR depth data per pixel

The generic formula for extracting distance from the depth data is given as follows (see Figure 26 for detail).

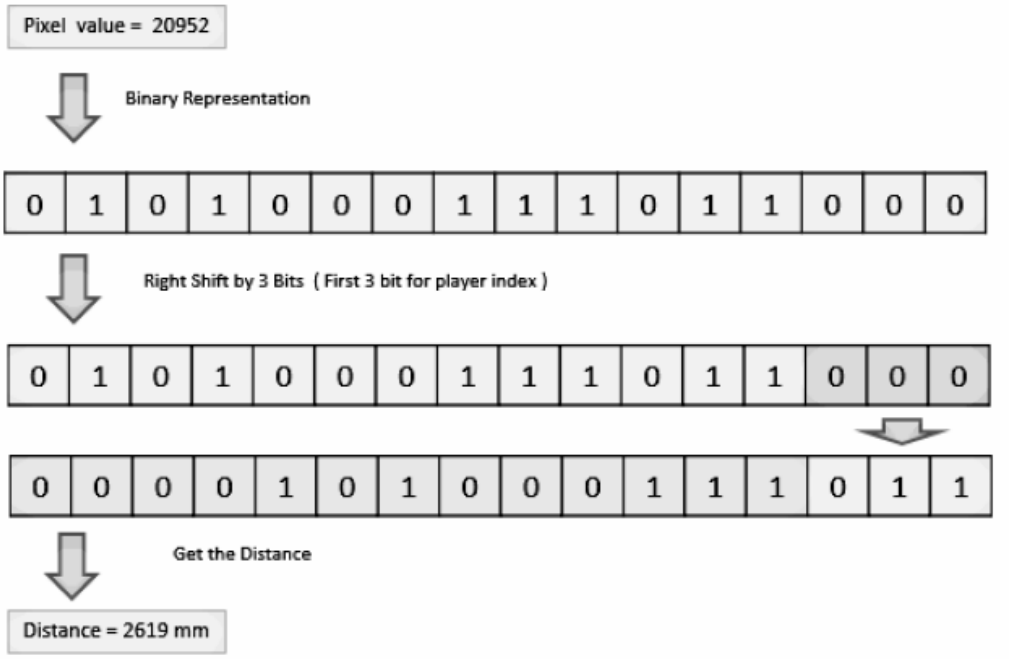


Figure 26. Sample code for depth calculation from IR depth data

The Microsoft Kinect for XBOX360's has a blind spot within the 80cm mark and beyond 8 meter. The range from 80cm to 8 meter is called "Default Range". Usually the most crucial information is right in front of the robot, especially within the 80 cm. Therefore, there is another mode called "Near Range" that is only present in Microsoft Kinect for Windows, as it has a shorter focal point with its additional integrated "Near Range Lens". To compensate for this blind spot, I added an external wide angle lens to reduce the blind spot range to exactly 40 cm, which will match the capability of the "Near Range", as shown in Figure 27.

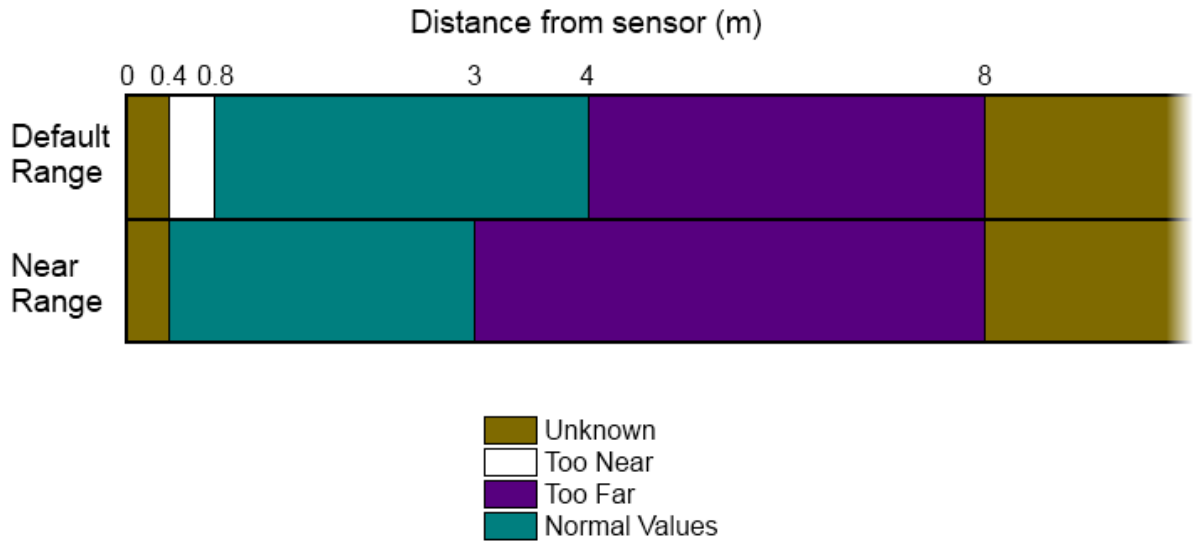


Figure 27. Kinect sensor depth range

In a typical application, Kinect's depth distance is broken in 4 catalogs: unknown, too near, too far and normal values. When an object is within the 80 cm mark, it is considered as too near, which means that the value is skewed. But there is a way to correct this by either using the built-in near lens in Kinect for Windows or applying an external wide angle lens. The normal value is a range between 80cm to 400cm or in near range mode the 40cm to 300cm. The return depth distance is directly corresponded to the real life distance. The too far value is between 400cm to 800cm, or in near range mode the 300cm to 800cm. When the object is within that range, the distance value from Kinect is inaccurate due to the disrupting of the IR laser and should not be trusted. And last, anything within the 40cm and beyond 800cm is unknown, where Kinect will not return any value.

Microsoft Kinect for Xbox360 does not have near range lens, hence has no near range mode. However, an external wide angle lens can be used, but an exponential function is required to correct the distance value that has been distorted by the wide angle lens.

The other type of data utilized for the mobile robot tank is the audio stream. Unlike the depth data, it does not need to be constantly streaming. This audio data acquisition is using a pooling mode with an event trigger.

When any noise greater than the ambient noise occurs, it triggers an audio sampling function. This function records an audio sample for a maximum of 4 seconds or until the noise drops to less than the ambient noise. This sample of audio clips is then used for further processing.

The triggering of sample recording is based on comparing the amplitude of ambient noise to sudden spikes of noise and the recording process begins when spike occurs.

Before any sample can be recorded, a `readyTimer` is initialized with `TimeSpan` 4 seconds. This step basically allocates a storage size for any recording of up to 4 seconds and the counter for steps is set to one at initialization of the program, as shown in Figure 28.

```
this.readyTimer = new DispatcherTimer();
this.readyTimer.Tick += this.ReadyTimerTick;
this.readyTimer.Interval = new TimeSpan(0, 0, 4);
this.readyTimer.Start();
```

Figure 28. Sample code of audio sample storage container

3.3.1.2 V2 Improvement

Color image processing is added to V2, which requires color data acquisition. Microsoft Kinect sensor is initialized by color data streaming with "ColorFramesReady" event trigger.

Microsoft Kinect sensor streams at 640x480x30fps with 3 color RGB profile plus a spacer (RGB: Red, Green, Blue and one bit of value = "0") in the form of 1D byte array. This color data array is up to 1228800 byte size or close to the marketing term: 1.3 Megapixels.

Each color pixel is arranged in the following fashion: Red, Green, Blue and a spacer. Each color value goes from 0 to 255 and on default the spacer value is "0". But, it can be utilized as an alpha value from 0 to 255, where "0" indicate no transparency/opacity and "255" referring to completely transparent.

3.3.2 Audio Signal Processing

The audio process breaks into 3 parts: filtering, processing and comparing. In the event that multiple voice commands are issued one after the other, there is a high case of computational congestion. This happens when a new voice command has been issued before the previous sample has yet to complete the audio process.

To prevent this, the audio process dynamically creates a new thread per audio sample and allows the audio acquisition to record the next sample while the previous sample is been processed.

Once the audio sample is recoded, it is filtered and corrected by a filtering system utilizing Acoustic Echo Cancellation (AEC), Noise Suppression (NS), and Automatic Gain Controller (AGC). At the same filtering system pipeline, the Bean Former (BF) and Sound Source Localizer (SSL) are implemented to create awareness of the sound source and can also make Kinect to listen/process to a sound signal from a particular direction.

When the sample enters the filter system pipeline, AEC is applied first to remove the sounds that are coming from loud speaker echo, NS is next used to suppress the undesirable background noise, and AGC

is utilized finally to enhance the sound amplitude for the next stage of audio signal processing, as shown in Figure 29.

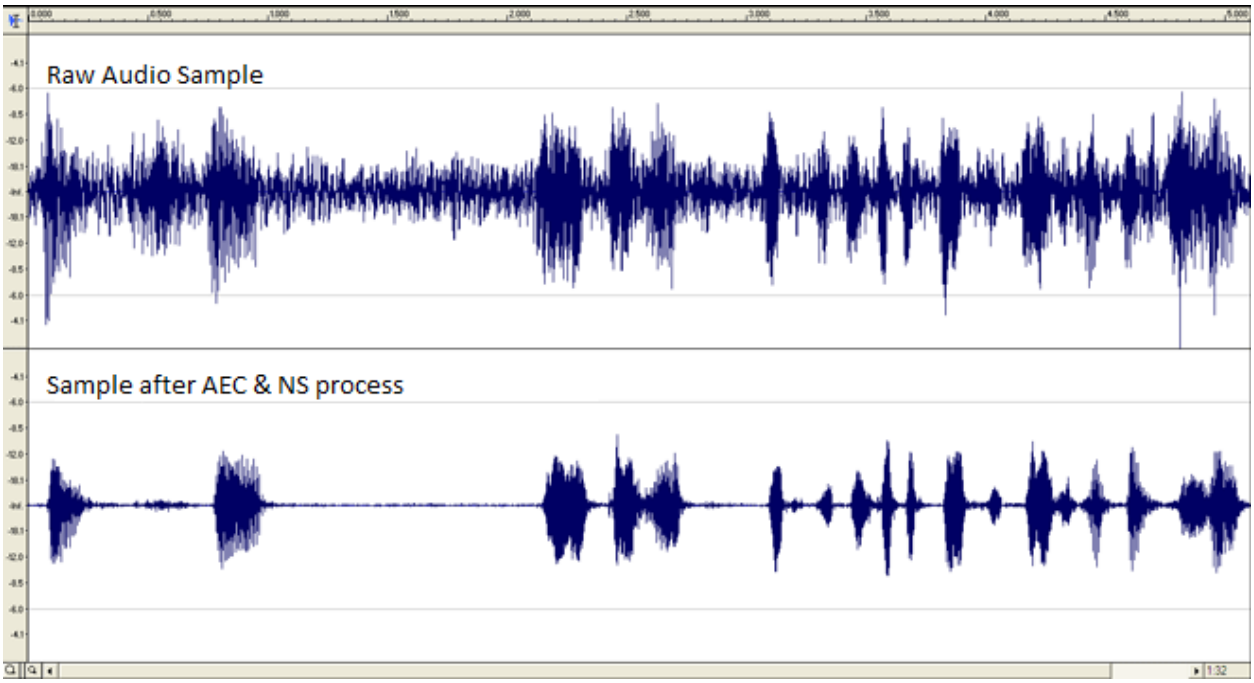


Figure 29. Acoustic echo cancellation and noise suppression process

Audio signal processing consists of 2 models: acoustic model and language model, as shown in Figure 30. First, the acoustic model analyzes the audio sample and converts it into a number of basic speech elements called “phonemes”. Second, the language model matches each individual phoneme with all the pronunciation template of alphabets and selects all of the alphabets whose probabilities are higher than the specified threshold. Once each phoneme has its own set of alphabets, the process combines those alphabets into words and computes the confidence level of each word from the possible alphabets.

Confidence level is a probability of each word based on the probability of the possible alphabets that were computed by the language model, in which, the percent of chances that the voice sample is corresponded to the word. The confidence level is set in the program and depends on the ambient noise and voice command issuer’s locale accent. The confidence level should be properly set.

The last step of the audio signal processing is to compare those computed words with a predefined library of words. There is also a high probability that multiple words match one predefined word in the library. In this case, the one with the highest probability will be chosen as the correlated word. However, this leads to a voice association error, such that if 2 words have similar wave form, the chance of misidentification is extremely high. For example, “Start” and “Stop” issued from a person with accent and with confidence of 70% will result in identical probability, meaning the misidentifying “Start” or “Stop” is extremely high.

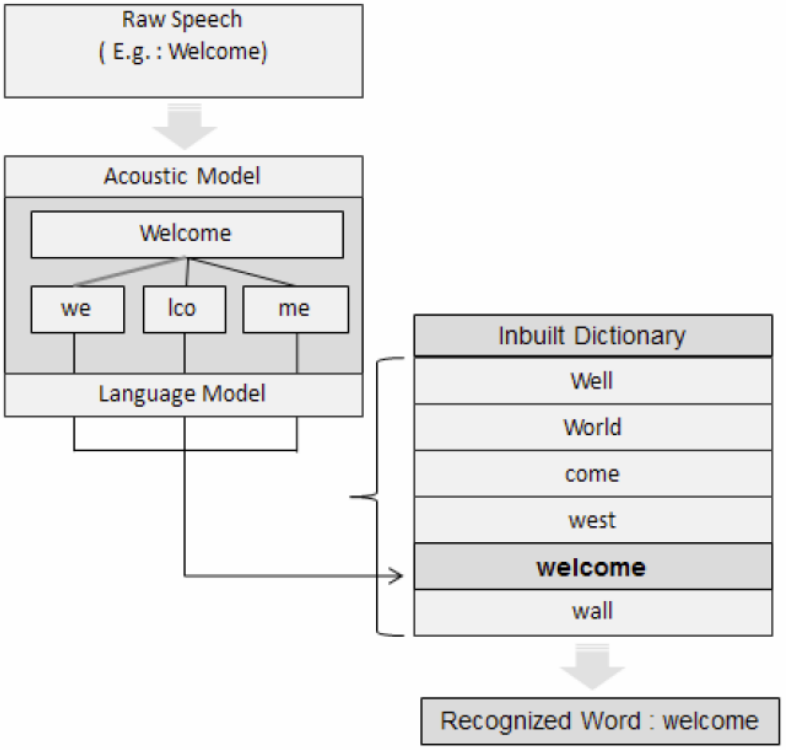


Figure 30. Phoneme to word process

The voice command can be either a single word or a sentence. Microsoft speech platform engine provides simple API for developer to tap into the speech system without much of mathematic and mechanistic background. In the case of sentence, the engine not only takes the word with the highest probability but also takes into consideration the probability of other lower probability words. In this case a device within the Microsoft speech platform engine, called “Grammar Builder”, is utilized.

Grammar builder breaks predefined sentences into word groups, as shown in Figure 31. There are few ways to use the grammar builder. The most common method is to lay out a sentence as it is. With this method, the grammar builder will use the sequences of each word as is, for example, red circle will be recognized but circle red will not. The other way is wild card method. In this case, sequences of words are not important, and selected keyword can be weighted much higher while the other words are lower. For example, for the sentence “draw a **RED** *Circle*”, “**RED**” has the highest weight and “Circle” has no sequence restriction and also has high weight, however, draw has the lowest weight and is therefore deemed non-critical. On the other hand, a phrase “Drain a Circle RED” will be identified as the proper phrasing sentence.

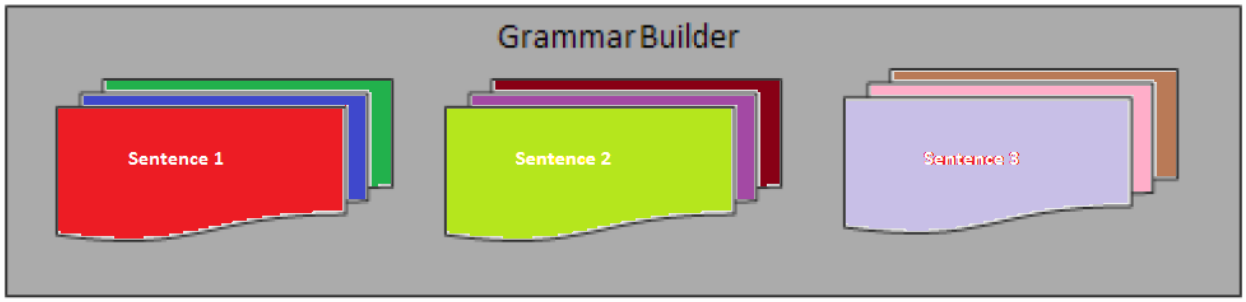


Figure 31. Common predefined sentences in grammar builder

In the mobile robot tank program, the sentence structure and word sequence is important to filter out errors due to the accent and noise. However, this requires more computational resources than the wild card mode.

Once the sample has gone through the audio signal processing pipeline and returns a valid selection, the selected voice command executes the appropriate logic function in the robot as shown in Table 5.

Table 5. Voice commands library

Voice Sample	Redefined Sentences/words	Functions
Robot Start	Robot Start	Manual Control = OFF Automatic Control = ON
Robot Backward	Robot Backward	Manual Control = ON Automatic Control = OFF PWM(0,1) = -30, -30
Robot Forward	Robot Forward	Manual Control = ON Automatic Control = OFF PWM(0,1) = 30, 30
Robot Turn Left	Robot Left	Manual Control = ON Automatic Control = OFF PWM(0,1) = 0, -15
Robot Turn Right	Robot Right	Manual Control = ON Automatic Control = OFF PWM(0,1) = -15, 0
Robot Stop	Robot Stop	Manual Control = ON Automatic Control = OFF PWM(0,1) = 0, 0
Robot Voice Halt	Voice Halt	Automatic Control = ON Ignore all voice commands unless = “Voice Resume”
Robot Voice Resume	Voice Halt	Automatic Control = ON Un-ignore all voice commands

3.3.3 Image Signal Processing

The robot tank control design is based on depth analysis, which means that the depth information is used as the primary input data for analysis and final decision. [37]

The following are the justifications for choosing depth information as input instead of color information:

1) Color image raw data information requires at least 3 times more data bandwidth due to the separated RBG color spectrum; while depth information is a single integer per pixel.
2) The majority of image processing filters and algorithms require that color image raw data should be converted to gray scale or YUV spectrum for further processing, so the conversion process adds additional complexity.
3) While in the gray scale different colors of object become hard to distinguish and identify, this leads to lots of false positive identification and errors.
4) Depth information provides the least amount of errors in regards to the position of any object and obstacle in comparison to stereo vision interpolation through RGB color image.

3.3.3.1 V1 Design

The image signal processing pipeline for the mobile robot tank breaks into 4 parts: depth data conversion, data processing, region of interest and logic command. The depth data acquired from raw data acquisition processing phase was in short array. This array can be copied into another physical storage container in the memory to free up the current image frame buffer and allow the next frame to overwrite the same address as soon as the copying is done. This also eliminates the possibility of data corruption due to reading and writing at the same time. The code for this process is shown in Figure 32.

```
depthPixels = new DepthImagePixel[307200];
depthFrame.CopyDepthImagePixelDataTo(depthPixels);
```

Figure 32. Code snippet - copying depth data to new container

However, in order to use OpenCV, a special Intel Image (iImage) format is utilized. The iImage format is a 2D array while the depth data is a 1D array. The code for converting the depth data to iImage is shown in Figure 33.The first column of iImage is Y, the second column is X, and the rest are data.

```
void dephPixelData2iImageGray(short[] depthData, Image<Gray, short> iImage)
{
    int x = 0;
    int y = 0;
    for (int i = 0; i < depthData.Length; i++)
    {
        x = i % 640;
        y = i / 640;
        iImage.Data[y, x, 0] = depthData[i];
    }

}
```

Figure 33. Code snippet - depth data to iImage

The same conversion has a GPU variance as well. However, due to the low resolution of the image, the conversion on GPU is ineffective, mainly due to the fixed initialization and data transfer process. If the resolution increases to 1920x1080 or higher, then the computation time exceeds the data transfer time between the GPU and CPU, so the GPU variance is ideal, as shown in **Table 6**. The process of initialization of data migration from CPU to GPU is fixed, where the GPU utilizes the full parallelism on all cores for the same process instead of serial data conversion.

Table 6. Data to iImage conversion time on CPU and GPU

Resolution Size	Data Size (bytes)		Time			
	RGB	Depth	RGB		Depth	
320 x 240	307200	76800	CPU	GPU	CPU	GPU
640 x 480	1228800	307200	<3 ms	10 ms	<3 ms	10 ms
800 x 600	1920000	480000	<3 ms	10 ms	<3 ms	10 ms
1024 x 768	3145728	786432	4 ms	10 ms	<3 ms	10 ms
1280 x 720	3686400	921600	4 ms	10 ms	<3 ms	10 ms
1280 x 1024	5242880	1310720	5 ms	10 ms	4 ms	10 ms
1400 x 900	5040000	1260000	5 ms	10 ms	4 ms	10 ms
1440 x 900	5184000	1296000	5 ms	10 ms	4 ms	10 ms
1600 x 900	5760000	1440000	7 ms	10 ms	4 ms	10 ms
1600 x 1080	6912000	1728000	9 ms	10 ms	4 ms	10 ms
1920 x 1080	8294400	2073600	11 ms	10 ms	5 ms	10 ms
1920 x 1200	9216000	2304000	11 ms	10 ms	5 ms	10 ms
2048 x 1152	9437184	2359296	11 ms	10 ms	5 ms	10 ms
2048 x 1600	13107200	3276800	12 ms	10 ms	5 ms	10 ms

The iImage is filtered through Gaussian smoothing filter to smooth out any errors in the pixels. The Gaussian smoothing function has 2 modes: fast (normalized boxed filter) and slow (Gaussian filter). Both modes can be used depending on data type. Depth image processing utilizes the slow mode, i.e. Gaussian filter due to the small data size in comparison to RGB image data.

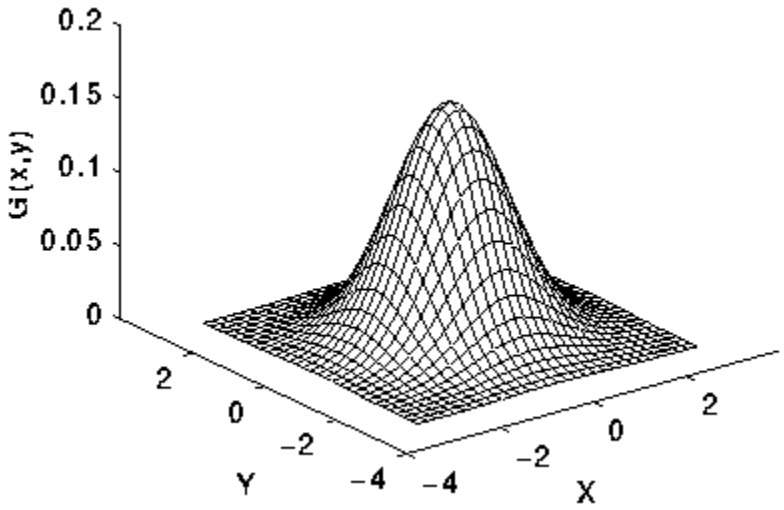
To perform a smoothing operation, a filter matrix is applied to the initial image. The most common filter is the linear filter in which a computed pixel value $g(i,j)$ is determined as the weighted sum of input pixel value $f(i+k,j+l)$ with weights $h(k,l)$, as shown in (3.1).

$$g(i,j) \quad = \sum_{k,l} f(i+k,j+l)h(k,l) \tag{3.1}$$

The fast mode in the smoothing process is the simplest filter matrix: each pixel output is the mean of its kernel neighbors, as shown in (3.2).

$$K = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \quad (3.2)$$

The slow mode in the smoothing process is the Gaussian filter. This is done by convolving each point of the initial image matrix with a Gaussian matrix, then summing it up to create a new output matrix, as shown in Figure 34. In the mobile robot tank program, a 5 x 5 Gaussian matrix is used.



$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\sigma = standard\ deviation\ of\ ditribution$$

$$in\ this\ case\ \sigma = 1$$

Figure 34. Gaussian 2D 5x5 Filter

After the smoothing is done, it feeds through the contour function of audio signal processing pipeline. The contour function is a GPU function, but a CPU version is also implemented as a fallback feature. The contour function searches for the points that are connected to a given point, by using a pre-set threshold to determine if the points are connected, as shown in Figure 35. Once all the connected points are found, it is stored to a buffer and counted as 1 object. This process continues until all the contours are found.

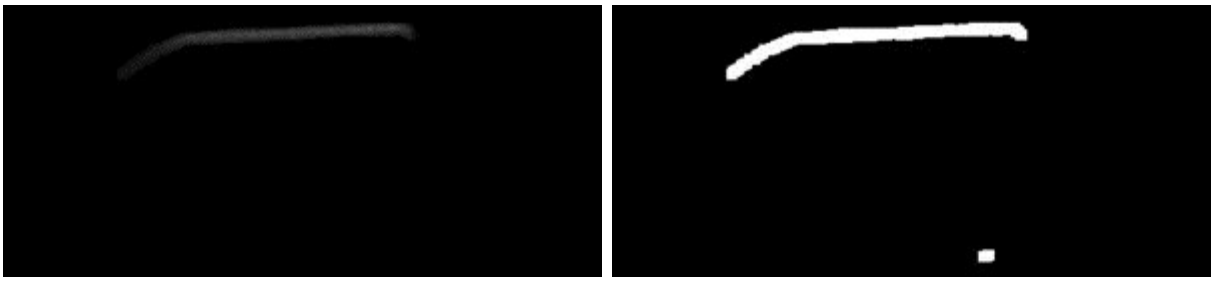


Figure 35. Original image (left) and contour image (right)

Contour process sometimes picks up clusters of noise as objects. Each object has its own size, called blob. The upper and lower limits of blob size are set to filter out those objects whose size is too small or too large, which effectively removes most of clusters of noise. During the filtering process, those acceptable blobs are recorded and a box is drawn around it to find out the maximum width and height. The same box is also being computed to determine the coordinate for the 4 corners. The size, 4 corner coordinates, and object number are used in the next phase for processing. The results from the contour process are shown in Figure 36.

Figure 36. Contour blobs, highlight boxes, and 4 corners

For the mobile robot tank to make its next move, it needs to identity if there is any obstacle in front. By definition, the only distance in question is just a few meters in front of the robot. This approach allows conservation of resources by only considering the lower half of the screen. Since the robot is only concerned with the front, the lower part of screen is also been divided into 3 parts. The middle part surrounded by a green square in Figure 37 is just big enough for the mobile robot tank to pass. The green square in Figure 37 is defined as Region of Interest (ROI).

During this phase, the ROI is constantly tested against the contour blobs by only comparing the 4 corners of each blob. Since a blob does not have fix shape, testing the exact edge of the blob against the

ROI is time consuming. However, test using the 4 corner points of the blob against the ROI is accurate enough.

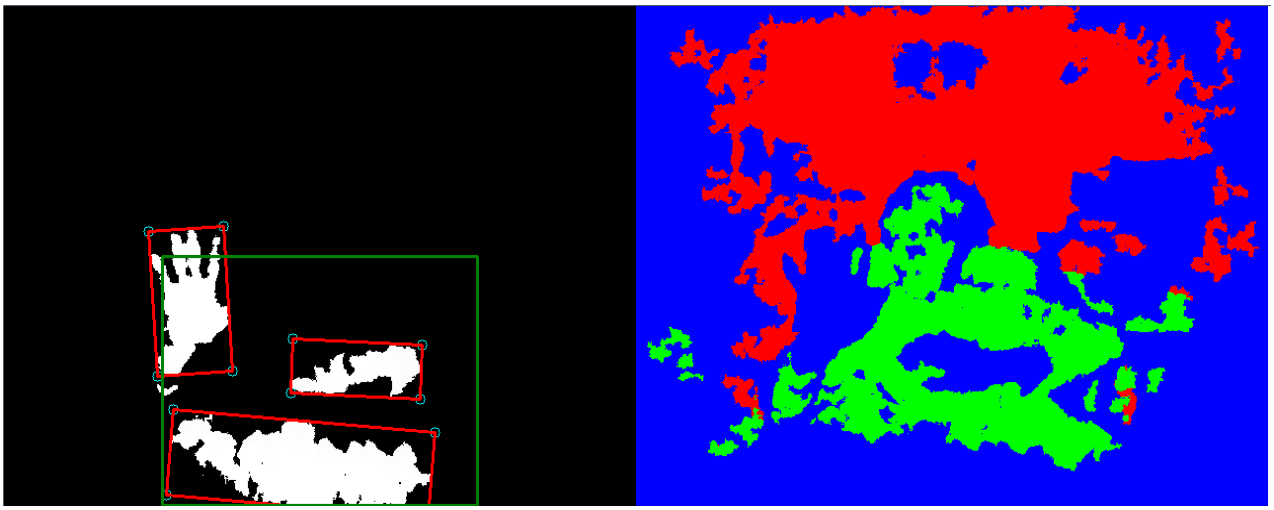


Figure 37. Region of interest (left), raw depth data (right)

When there is no object in the ROI, the robot issues a logic command “Go Forward”. If there is an object in the ROI and the right side of ROI, but the left to the ROI is clear, the robot issues “Turn Left”. If there is an object in the ROI and the left side of ROI, but the right to the ROI is clear, the robot issues “Turn right”. If the front, left side, and right side of the ROI have objects, the robot first sums up the total areas of the blobs from the center of the view all the way to left and right, and then compares two areas to determine which direction it turns. The robot turns to the side with a lower total area of the blobs to rescan an image and repeat the process again. In rare chances the left and right side total areas of the blobs are the same, the priority is to turn right. If the robot cannot find a clear path to go after 10 consecutive turnings, it is deemed no way out so the robot issues logic command of “No Way” to the communication signal processing. The logic commands are listed in Table 7.

Table 7. ROI conditions and logic commands

Left of ROI	ROI	Right of ROI	Logic Command
Object(s)	Clear	Object(s)	Go Forward
Object(s)	Clear	Clear	Go Forward
Clear	Clear	Object(s)	Go Forward
Object(s)	Object(s)	Clear	Turn Right
Clear	Object(s)	Object(s)	Turn Left
Clear	Object(s)	Clear	Turn Right
Object(s)	Object(s)	Object(s)	If (Left Total Blob Area > Right Total Blob Area) ➔ Turn Right
Object(s)	Object(s)	Object(s)	If (Left Total Blob Area < Right Total Blob Area) ➔ Turn Left

Object(s)	Object(s)	Object(s)	If (Left Total Blob Area = Right Total Blob Area) ➔ Turn Right
Object(s)	Object(s)	Object(s)	If ((Left Turns + Right Turns) > 10) ➔ No Way

3.3.3.2 V2 Improvement

Color image processing capability is added to V2 to improve object recognition and obstacle avoidance with the assistance of depth data processing. The color data from Microsoft Kinect is different from OpenCV iImage format. The data format from Microsoft Kinect Senor is composed of 4 bytes per pixel in a 1D array (Red, Green, Blue, and Spacer) while OpenCV iImage is a 2D byte array with 3 bytes per pixel (Blue, Green, and Red) and is arrangeed in the [Y,X] orientation instead of traditional [X,Y] axis. To ensure future scalability and different resolution, a dynamic premier is added to define the possible different image width and height. The code for this is shown in Figure 38.

```
void colorPixelData2iImageColor(byte[] colorDataByte, Image<Bgr, byte> iImage, int
imageWidth, int imageHeight)
{
    for (int y = 0; y < imageHeight; y++)
    {
        for (int x = 0; x < imageWidth; x++)
        {
            iImage.Data[y,x,0]=colorDataByte[y*imageWidth*4+x*4];// Red -> Blue
            iImage.Data[y,x,1]=colorDataByte[y*imageWidth*4+x*4+1];// Geen -> Green
            iImage.Data[y,x,2]=colorDataByte[y*imageWidth*4+x*4+2];//  Blue -> Red
        }
    }
}
```

Figure 38. Code snippet - color data to iImage

Unlike V1, the ROI of V2 is set to a default of 5x4 grids instead of only 1 in V1 and is scalable to any other configuration. This ROI can be used for highlighting and detecting objects for both color image processing and depth data processing, as shown in Figure 39.

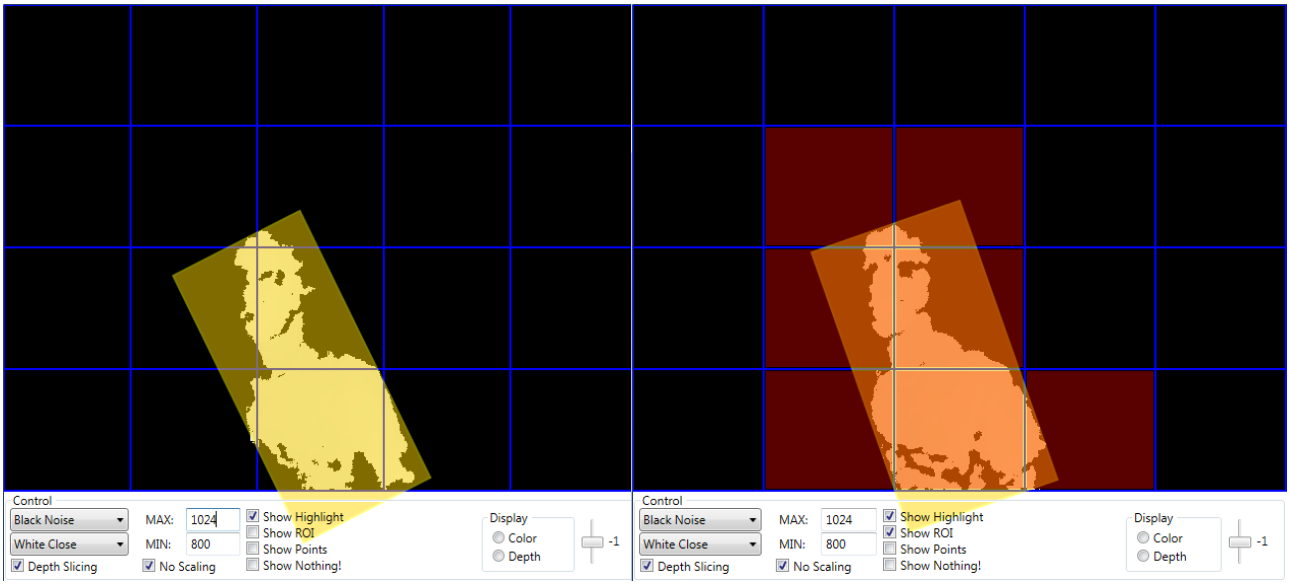


Figure 39. V2 depth data processing: object highlighting (left) and ROI (right)

The depth data processing of ROI in V2 inherits the foundation design from V1, but has improved on the detection speed, plus a new feature: object merging ability which combines multiple objects into one when they fit within the threshold, as shown in Figure 40.

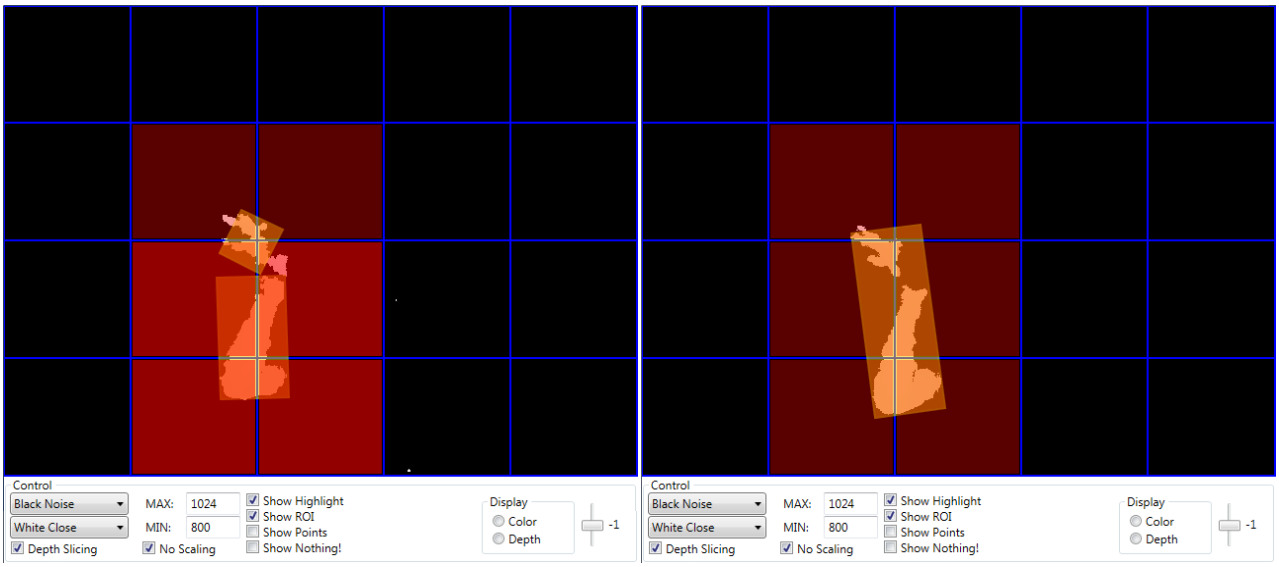


Figure 40. V2 depth data processing: multiple object detection (left) and multiple object merging (right)

The color image detection was tested and implemented using the following algorithms: Scale-Invariant Feature Transform (SIFT), Speed Up Robust Features (SURFT), Feature from Accelerated Segment Test (FAST) and Histogram of Oriented Gradient (HOG). All four algorithms have both GPU and CPU implementation variances.

In order to have a proper benchmark in performance, several criteria are set to achieve comparable results. The preset criteria are as follows: the maximum number of detected features is set to 500 and the uniqueness threshold is set to 80%.

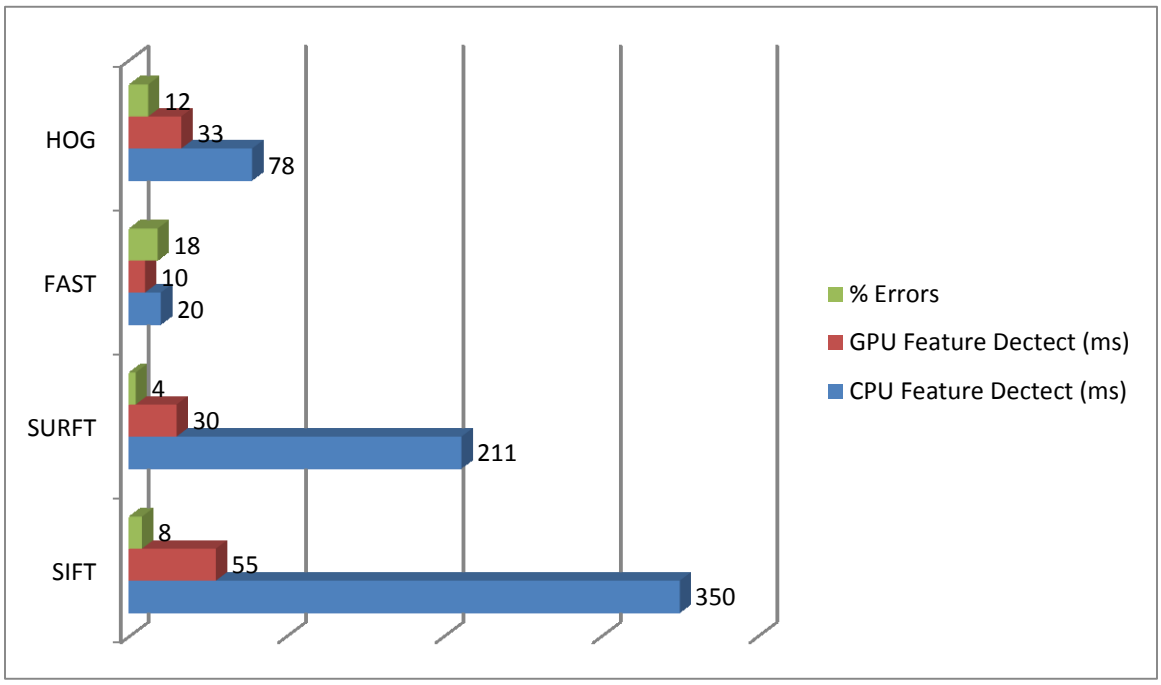


Figure 41. Comparison of different algorithms

Histogram of Oriented Gradients (HOG): This detection algorithm utilizes the gradient on the image, which provides edge information. The essential thought behind the HOG descriptors is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions. The implementation of these descriptors can be achieved by dividing the image into small connected regions, called cells, and for each cell we gather histogram of gradient directions and edge orientations for the pixels within the cell. The combination of these histograms then represents the descriptor. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization provides identification of changes in illumination or shadowing.

Fast Accelerated Segment Test (FAST): FAST, namely a corner detector, uses a circle of 16 pixels (a Bresenham circle of radius 3) to classify whether a candidate point p is actually a corner. Each pixel in the circle is labeled with an integer number 1 to 16 clockwise. If a set of N continuous pixels in the circle are all brighter than the intensity of candidate pixel p (denoted by I(p)) plus a threshold value T or all darker than the intensity of candidate pixel p minus threshold value T, then p is classified as a corner. The conditions can be written as:

Condition 1: I(x) >I(p)+T for all x in a set of N continuous pixels S;

Condition 2: I(x) <I(p)-T for all x in a set of N continuous pixels S.

So when either of the two conditions is met, candidate p can be classified as a corner. There is a tradeoff of choosing N, the number of continuous pixels and the threshold value T. On one hand the number of detected corner points should not be too many, on the other hand, the high performance should not be achieved by sacrificing computational efficiency. Without the improvement of machine learning, N is usually chosen as 12. A high-speed test method could be applied to exclude non-corner points.

Speed Up Robust Features (SURF): It uses an integer approximation to the determinant of Hessian blob detector, which can be computed extremely fast with an integral image (3 integer operations). For features, it uses the sum of the Haar wavelet response around the point of interest. Again, these can be computed with the aid of the integral image.

Scale-Invariant Feature Transformation (SIFT): For any object in an image, interesting points on the object can be extracted to provide a feature description of the object. This description, extracted from a training image, can then be used to identify the object when attempting to locate the object in a test image containing many other objects. To perform reliable recognition, it is important that the features extracted from the training image be detectable even under changes in image scale, noise and illumination. Such points usually lie on high-contrast regions of the image, such as object edges.

FAST algorithm provides the fastest processing speed but has the most errors. The SURFT provides the most robust detection but at the cost of processing time. However, HOG varies in detection error due to the histogram detection of the image based on the light distribution. The comparison among these algorithms is shown in Figure 41 .

To achieve object recognition from color data, I constructed two methods of detection: Quick Respond (QR) code recognition and traffic sign recognition. QR code was developed for the automobile industry for fast bar code recognition for parts tracking during the manufacturing process. QR code is a 2D bar code with no fixed orientation. However, due to the complexity of the black block arrangement (see Figure 42), a clear image is needed for QR processing. Microsoft Kinect sensor does not provide a sharp enough image for recognition due to the fixed lens aperture. In other words, the Kinect sensor can only capture a clear enough image for QR recognition when the QR code is within 80 cm or less. This distance is exactly at the cut off minimum range of the depth data. And the QR code recognition based on the Google ZX library does not provide information for object orientation. The QR code recognition is shown in Figure 42.

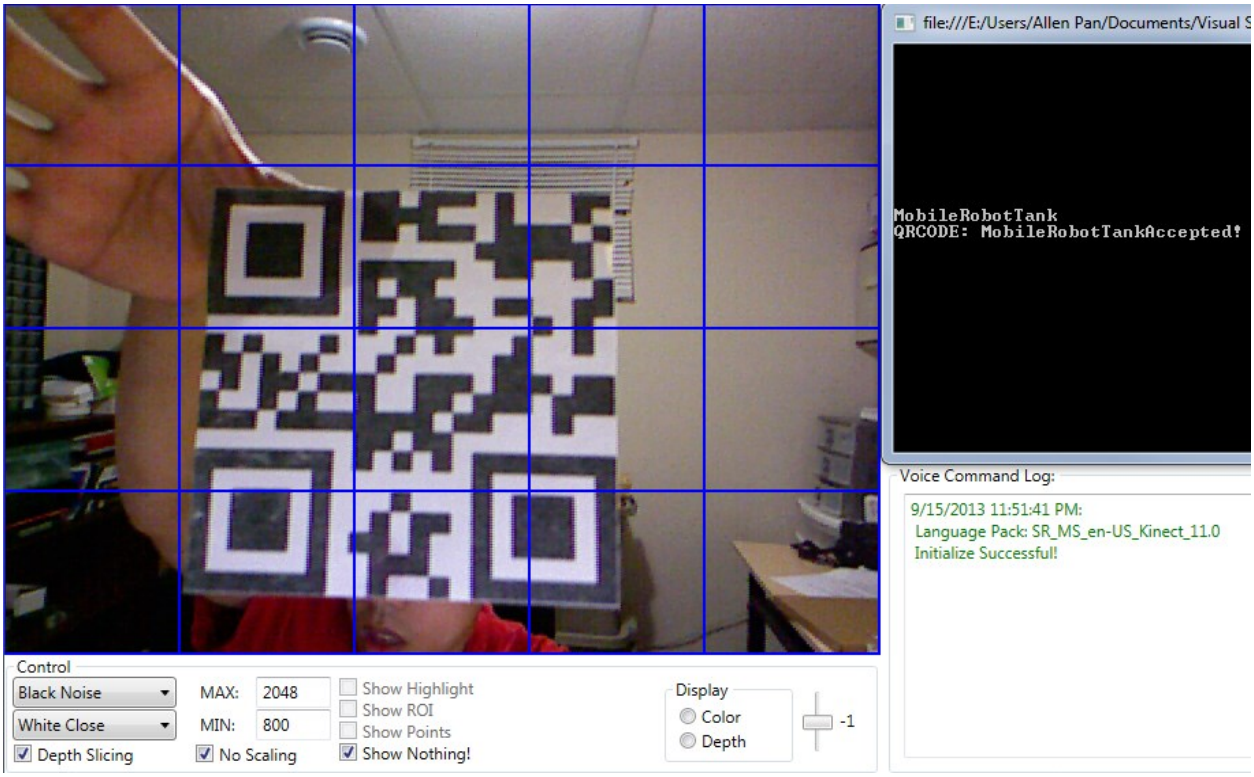


Figure 42. QR code recognition with sample code

This leads to the second object recognition method by using traffic signs as a template. The mentality of utilizing traffic signs as recognizable objects is due to the distinctiveness and unique features of each individual signs, which allows an easier way to construct and manage a template library.

In this V2 program, “STOP” sign template feature training is addressed in the library by the following distinctive features: hexagon shape, red background, and white text with the word “STOP”, as shown in Figure 43.

One of the major problems with the current sign recognition is lack of glare filtering. In this case, the upper STOP sign is not recognized due to the excess of light glare. This can be improved by implementing HOG algorithm.

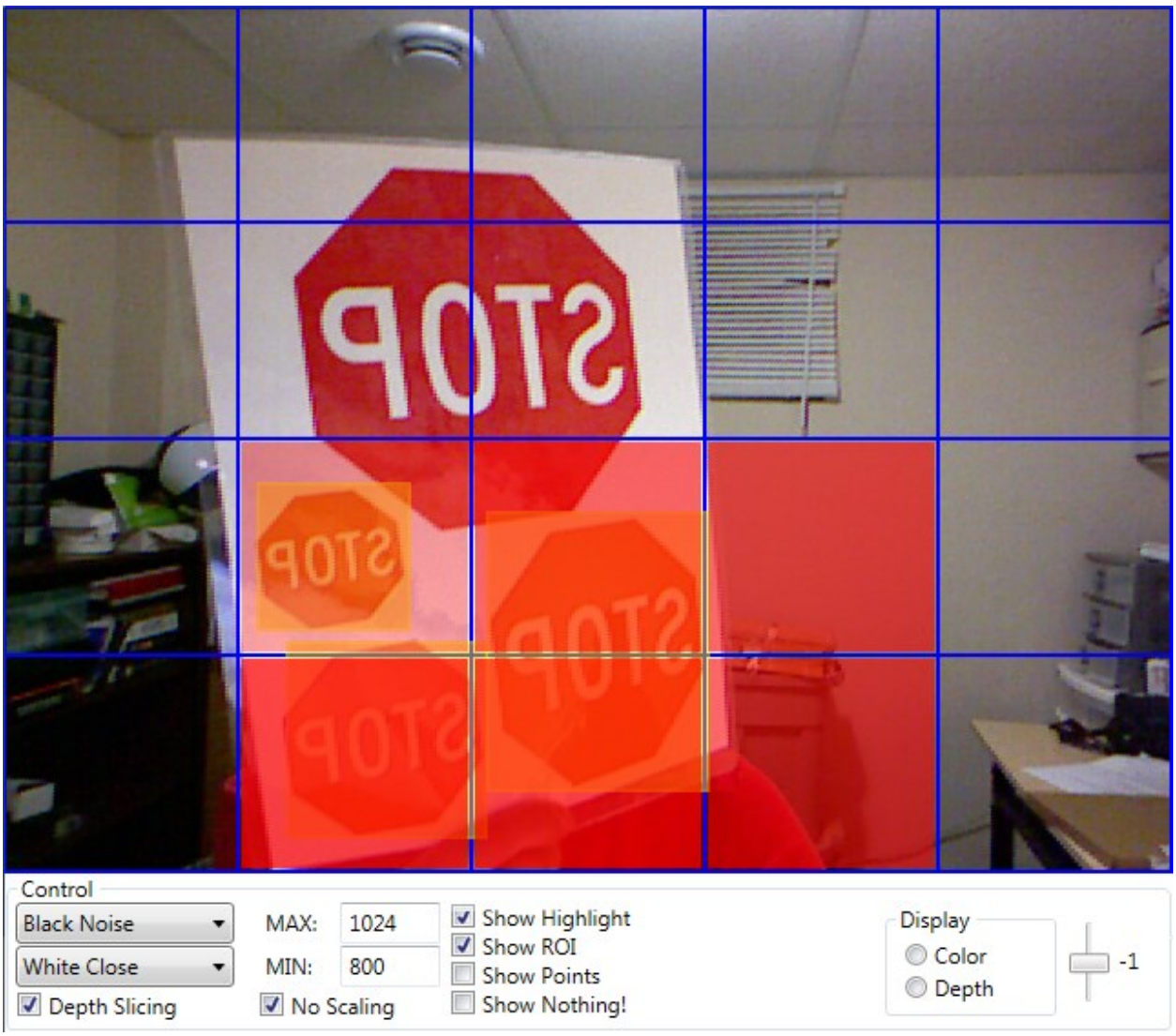


Figure 43. Traffic sign recognition (stop sign)

The logic process in the V1 has been expanded to compensate the ROI detection. Therefore, the logic process in V2 has increased to 5 choices which translate to: far left, left, center, right and far right. The logic process is as follows.

When the “STOP” sign is detected within a distance greater than 1024 mm and less than 2048 mm, the traffic sign detection function determines the occupied column. If the occupied column is not the center column, it first rotates the mobile robot tank until the sign is in the center. Next it compares with the empty space from blob detection. If the center is empty, then the program generates the command “go forward”. If the center is not empty, the mobile robot tank keeps correcting itself until the center is free and then goes forward. This cycle continue until the sign is less than 1024 mm or greater than 2048 mm and then the mobile robot tank comes to a halt. However, if the forward command has not been ever issued within 30 seconds, the mobile robot tank will determine if there is no way out. In this situation,

the mobile robot tank will stop and time out for 1 minute. This process repeats until the mobile robot tank powers off, as shown in Figure 44.

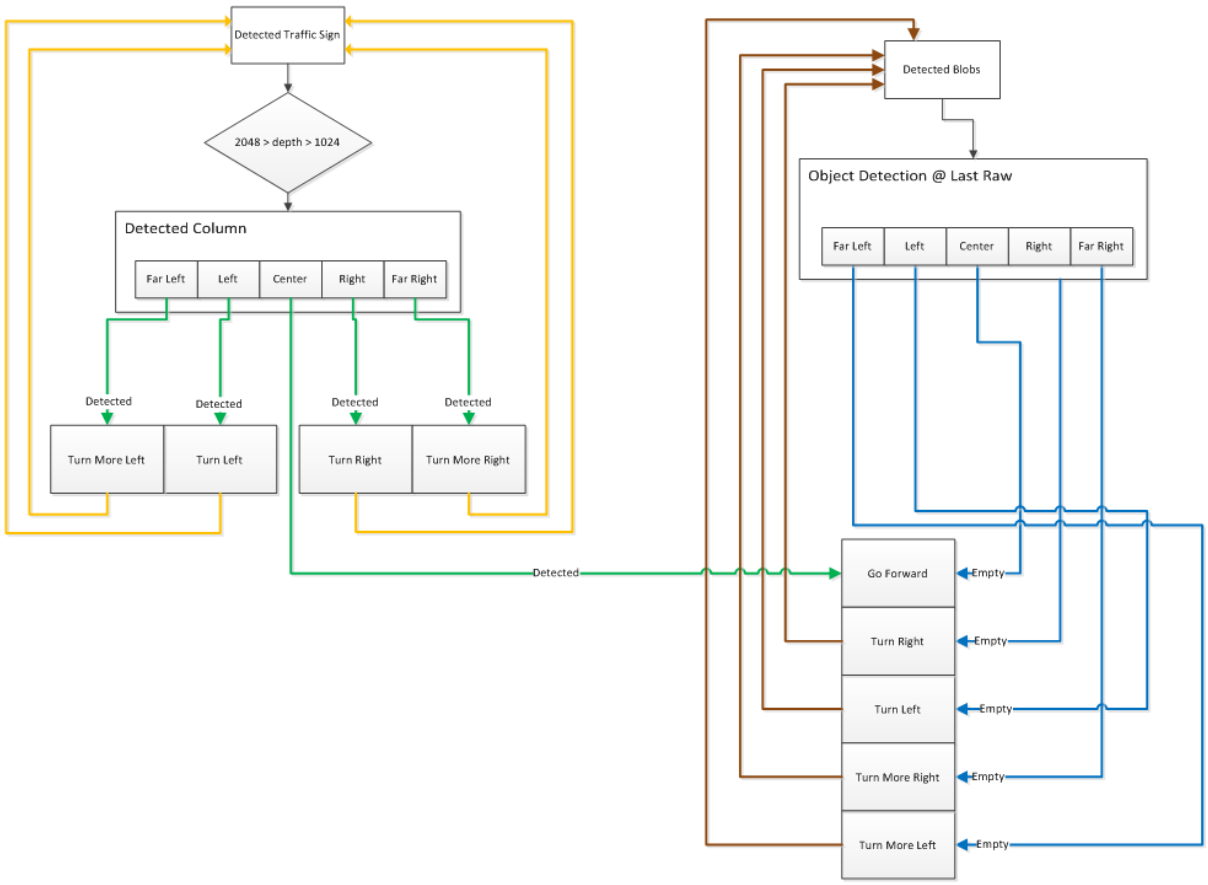


Figure 44. Mobile robot tank movement logic

3.3.4 Communication Signal Processing

The very last phase of the robot software design is converting the logic commands to movements. This is done through the communication protocol. The communication signal processing is implemented through a serial com port between HPC (High Performance Computer) and TI DSP Controller by using RS232. This communication process is done in a few different stages and in every stage it will send an array of 16 bytes.

This byte array is structured as the following: 1st byte is a character (U = Upper Limit, L = Lower Limit, S = Start, P = Stop, R=Run), the 2nd byte is the upper 8 bits of PWM0, 3rd bye is the lower 8bits of PWM0, the 4th byte is the upper 8 bits of PWM1, 5th bye is the lower 8bits of PWM1, the 6th byte is the upper 8 bits of PWM2, 7th bye is the lower 8bits of PWM2, the 8th byte is the upper 8 bits of PWM3, 9th bye is the lower 8bits of PWM3, the 10th byte is the upper 8 bits of PWM4, 11th bye is the lower 8bits of PWM4, the 12th byte is the upper 8 bits of PWM5, 13th bye is the lower 8bits of PWM5, the 14th byte is the timeout, the 15h byte is the upper 8 bits of CRC code, 16th byte is the lower 8bits of CRC code, as shown in **Table 8**.

Table 8. Communication byte array

Byte Array Number	
[0]	Character (U, L, S, P)
[1]	PWM0 upper 8 bits
[2]	PWM0 lower 8 bits
[3]	PWM1 upper 8 bits
[4]	PWM1 lower 8 bits
[5]	PWM2 upper 8 bits
[6]	PWM2 lower 8 bits
[7]	PWM3 upper 8 bits
[8]	PWM3 lower 8 bits
[9]	PWM4 upper 8 bits
[10]	PWM4 lower 8 bits
[11]	PWM5 upper 8 bits
[12]	PWM5 lower 8 bits
[13]	Timeout
[14]	CRC upper 8 bits
[15]	CRC lower 8 bits

The RS232 protocol was designed for other robots, so for the mobile robot tank, “U” and “L” are not used and the values for PWM0-5 should be set to the maximum 0xFFFF and the minimum 0x0000, respectively. When the DSP receives an array of 16 bytes, it checks the command code first. The DSP interprets the values for PWM0-5 as the upper limit for the command code “U” and the lower limit for “L”. As soon as receives an array with “S”, the DSP will start the DSP timer, wait for the idle time, activate the relay which powers the motor driver circuit, and the robot is ready to move. After that, the DSP will receive an array with “R” and send PWMs that comes with “R” to the motor driver circuit to run the robot. As soon as receives an array with “P”, the DSP will stop the DSP timer and de-activate the relay which powers off the motor driver circuit and stops the robot tank.

3.4 Hardware Design and Implementation

The hardware design of the mobile robot tank consists of: power regulation/power distribution system, low power high performance computer, Single Board Computer (SBC) system.

3.4.1 Power Regulation and Power Distribution System

There are diverse components that are being used in the mobile robot tank, which creates a challenge to manage the consistency and diversity of all the different power requirements. Table 9 shows the requirements for power supplies of the mobile robot tank.

Table 9. Mobile robot tank power requirement chart

Voltage	Total Amperage	Components
3.3V	0.5 A	Ti DSP Core
5V	2 A	DSP Board, Kinect Sensor, ARM SBC
12V	1.25 A	Kinect Sensor

19V	4 A	HPC
12V -24V	2 A	H-Bridge, Motors

To simplify the power regulation design, a single voltage source with sufficient amperage should be adequate to power the entire mobile robot tank. During the initial prototyping stage, the most demanding part of the mobile robot tank is the HPC which requires a supply of 19V with 4A. Since the HPC maximum draw is 6.25A, there is still a lot of overhead for powering the rest of the system.

3.4.1.1 V1 Design – Linear Regulation

In the lab test, it shows that H-bridge and motors are capable of running as low as 12V and as high as 24V, which makes the regulation circuit much easier, as the only 2 voltage levels are needed: 12V and 5V. To ensure stability and consistency, two 12V 3A and two 5V 3A rails are implemented as shown in Figure 45.

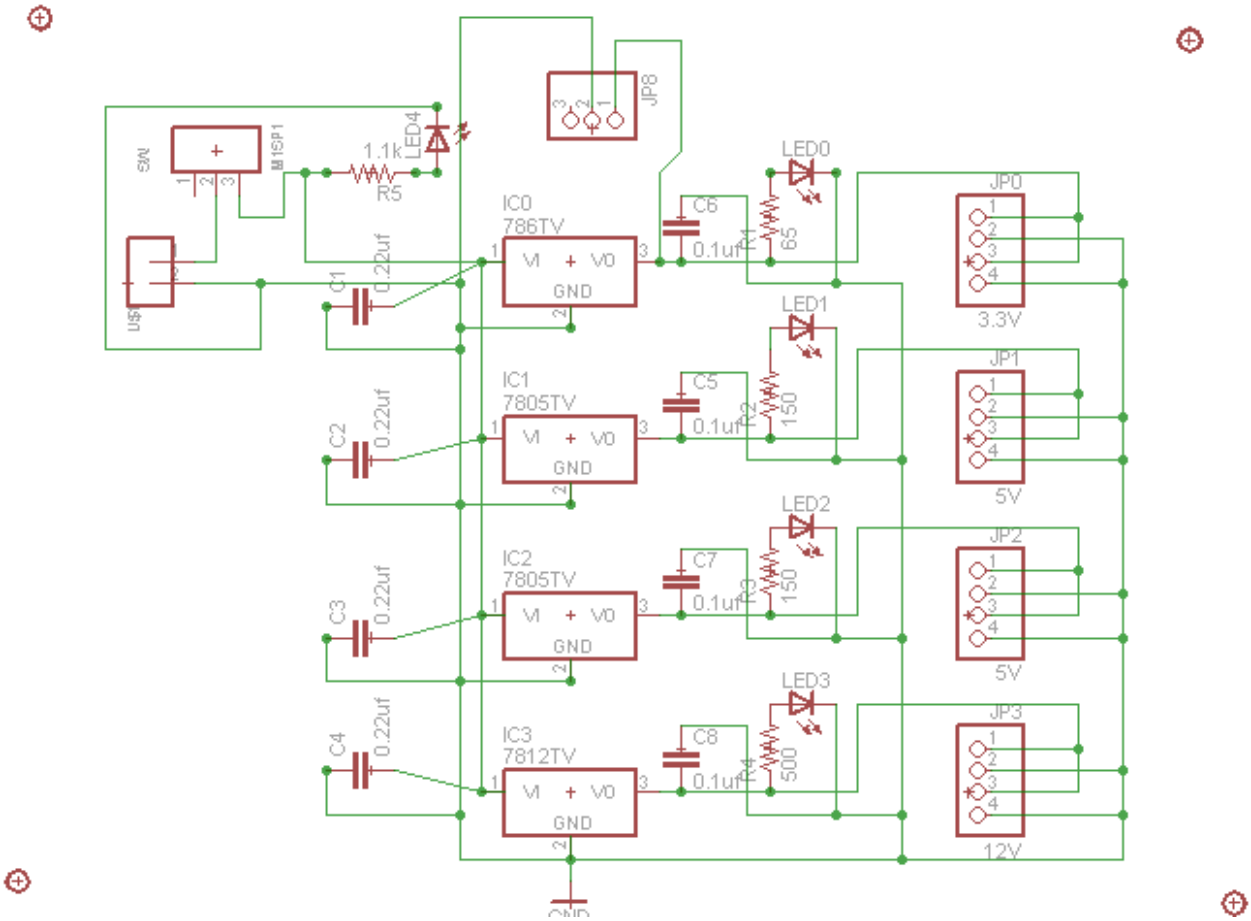


Figure 45. V1 and V2 power circuit schematic

This design utilizes linear voltage regulation ICs: LT1805CT-5 and LT1805CT-12. LT1805CT is a fixed voltage 3A linear regulated integrated circuit. Moreover, linear voltage regulation usually has lower

efficiency in comparison to switching regulating circuit. This was not fully discovered until the first prototype-board was built and even at the no load, the linear voltage regulator exceeded more than 80^{o}C without any heat sink, and as a result, the thermal protection is triggered.

A simple low profile heat sink is attached, which was created from an aluminum socket 462 CPU heat sink. With this addition, the temperature on the regulator with no load reduced to 40^{o}C, but as soon as a moderate load is connected, the temperature would exceed 80^{o}C, and once again the thermal protection is triggered.

No matter what orientation of the heat sink is placed, it always triggers the thermal protection. An active cooling solution is required. A simple 40mm ball baring fan was added perpendicularly to the heat sink to ensure fast heat relocation and stability of the temperature normalized at 60^{o}C at full loads.

However, this prototype board design raises a question on stack ability and thermal dynamics. Since the power regulation circuit is enclosed in a robot with limited space and it has minimum air exchange, a different type of layout of the components is needed. While the vertical mounted fan does its job as an active cooling solution; however, within the robot body, there is height restriction to prevent me to utilize a standard ball baring fan.

The prototype board design utilizes an axial-flow fan, which moves air parallel to the shaft of the rotational blade. In order to have air flow, the fan must be mounted vertically to the heat sink. A centrifuge fan allows air flow to be perpendicularly from the shaft, which allows the fan to be mounted parallel to the PCB. The centrifugal fan is also a common choice of active cooling for small form factor device where the free space is limited.

3.4.1.2 V2 Design – Linear Regulation

The V2 power regulation circuit design focuses on the foot print, stack ability, and long term stability. The foot prints and stack ability goes hand in hand. The DSP board was designed with specific mounting holes and dimension, which means that the V2 design must be following exactly the same dimension and mounting requirement, see Figure 46. Since the V2 will be stacking along with other boards in a closed environment, the proper air circulation to prevent overheating is crucial to the design.

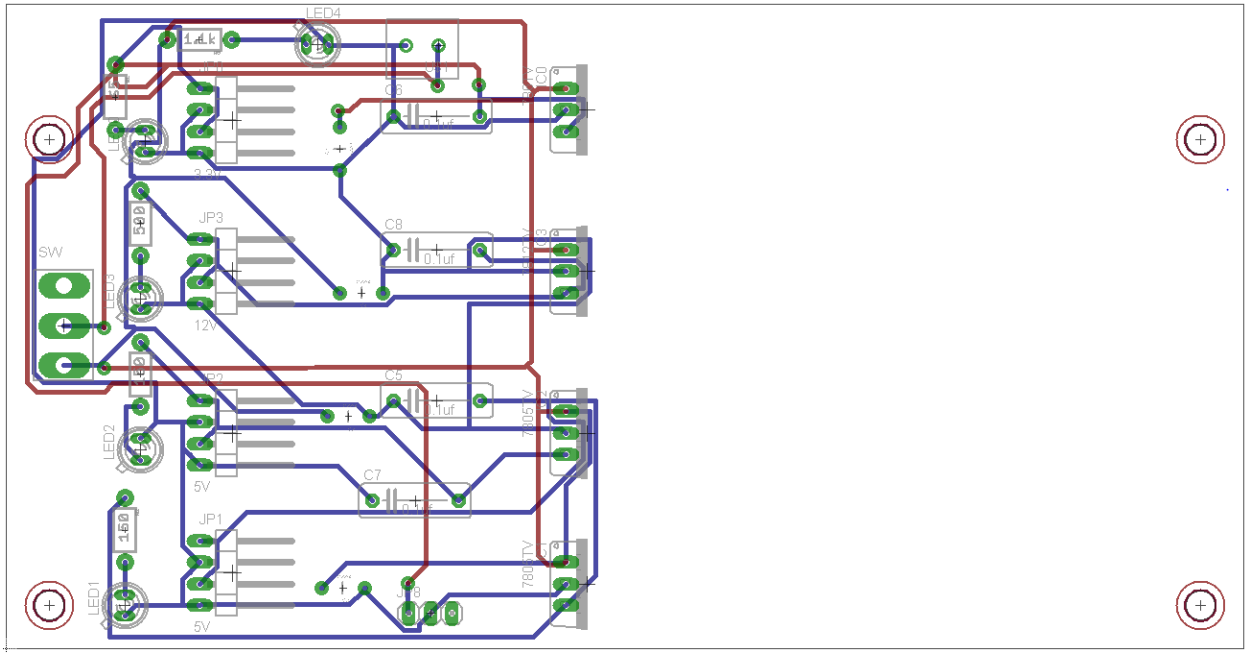


Figure 46. V2 power circuit PCB

V2 PCB board is utilizing the same voltage regulator but with a higher voltage solid state capacitor to ensure stability in high temperature operation. The heat sink has also been scaled up by utilizing an Intel Pentium 4 retail heat sink which provides 22 larger aluminum fins in comparison to 16 smaller black painted aluminum fins.

The active cooling calls for a space effective solution: a centrifuge fan that is bundled with ASUS motherboard for PowerMOS (manufactured by: Y.S. Tech – YD124515MB, 12V 0.15A). The centrifuge fan channels the air in from the top and the bottom of the PCB to cool the regulators where they are mounted on a Pentium 4 heat sink base.

The choice of fan is centrifuge due to the limited space on the PCB and the air direction. Regular ball bearing fan has vertical air access with cold air intake from top and hot air exit from bottom. Due to the vertical mounting nature of voltage regular and heat sink, the heat sink fin requires cool air to pass parallel to the PCB. Centrifuge fan allows cool air intake from both top and bottom of vertical axis and hot air exits at the horizontal axis, in which cool down the voltage regulator heat sink.

3.4.1.3 V3 Design – Switching Regulation

Linear regulator has lower efficiency than switching regulator, hence, less heat dissipation, lower operating temperature and simpler cooling solution. On the other hand, switching regulation circuit is more complex to design than the linear regulation circuit and requires more components and the proper coupled capacitor to achieve the proper switching frequency.

The switching regulator LM2596, with stable output of 3A load current, is manufactured by Texas Instrument. The 12V switching regulator is LM2596-12, while the 5V is LM2596-5. The LM2596 has very strict component requirements when it comes to capacitor and inductor couplers, refer to **Table 10**.

Table 10. LM2596 fixed voltage component selection table

Conditions			Inductors		Capacitors			
					Panasonic HFQ	Nichicon PL	AVTX TPS	Sprague 595D
V_{out}	I_{out}	V_{in}	µH	L Type	µF/V	µF/V	µF/V	µF/V
5V	3A	8	22	L41	470/25	560/16	220/10	330/10
		10	22	L41	560/25	560/25	220/10	330/10
		15	33	L40	330/35	330/35	220/10	330/10
		40	47	L39	330/35	270/35	220/10	330/10
12V	3A	15	22	L41	470/25	470/25	100/16	180/16
		18	33	L40	330/25	330/25	100/16	180/16
		30	68	L44	180/25	180/25	100/16	120/20
		40	68	L44	180/35	180/35	100/16	120/20

In addition to the restricted component selections, the majority of Through Hole Devices (THD) has been replaced with Surface Mount Device (SMD). This strategic component specification change is to ensure higher density of components on the same PCB foot print and frees up additional real-estate as cooling plane. SMD components are generally smaller and cheaper, which results in lower production cost and fully utilizable double sided PCB. The schematics for V3 design is shown in Figure 47.

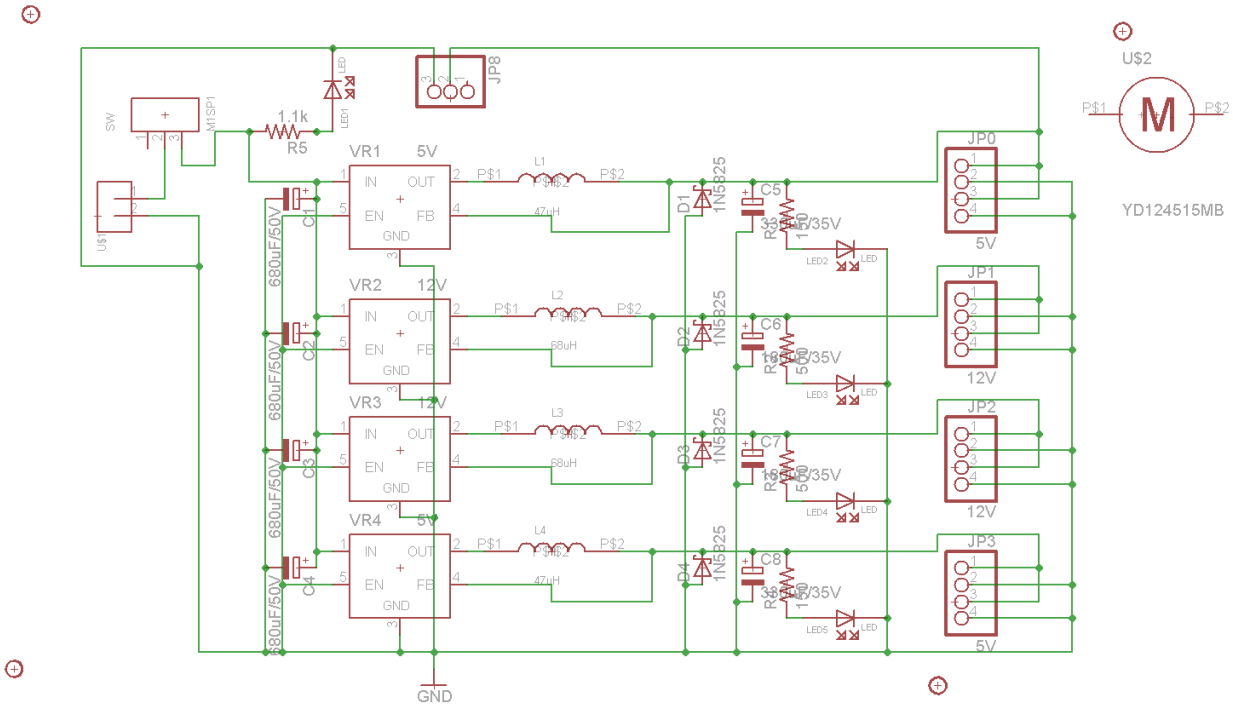


Figure 47. V3 power circuit schematic

The V3 PCB also improved the cooling as well. An air well has been integrated with the PCB and as a result, the dual air intakes from top and bottom of PCB. 2 more mounting holes were used for securing the centrifuge fan, as shown in Figure 48.

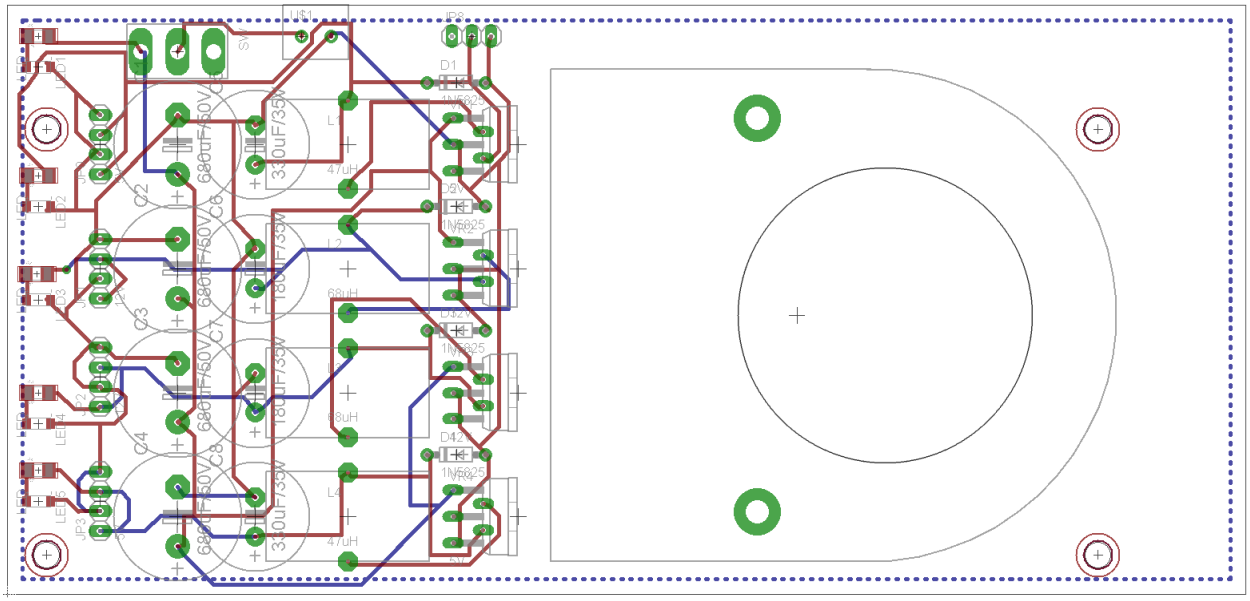


Figure 48. V3 power circuit PCB

However, V3 power circuit was unable to perform to the required specification. Each rail's powered LED flickers a lot, which is an indication of unstable power and improper switching frequency. Oscilloscope shows a large ripple from output voltage (V_{out}) when 19V is applied at voltage input (V_{in}). I attempted to correct such swing by increasing the capacitor voltage rating, and it shows no effect. However, by increasing the capacitance value, this shows less frequency blinking and lesser swing. The result of V3 design is unable to provide enough current to the whole system.

It shows no much improvement after additional part replacement and it has come to my attention that the PCB design is at fault. Reference data sheet DCDCCONV provided by Jaycar Electronic shows some light on the design fault [38]. Linear technology application note on switching regulator LTC3536 [39] shows the following flaws in the PCB routing and design:

1) There is no dedicated ground plane for each rail.
2) The parasitic inductance and resistance of all circulating high current paths should be minimized. This can be accomplished by keeping the routes as short and as wide as possible. Capacitor ground connections should via down to the ground plane by way of the shortest route possible. The bypass capacitors on V_{IN} and V_{OUT} should be placed as close to the IC as possible and should have the shortest possible paths to ground.
3) The exposed pad is the electrical power ground connection for the switching regulator in the DD package. Multiple vias should connect the backpad directly to the ground plane. In addition,

maximization of the metallization connected to the backpad will improve the thermal environment and improve the power handling capabilities of the IC in either package.

4) The components with high current should all be placed over a complete ground plane to minimize loop cross-sectional areas. This minimizes electromagnetic interference and reduces inductive drops.
5) Connections to all of the components with high current should be made as wide as possible to reduce the series resistance. This will improve efficiency and maximize the output current capability of the buck-boost converter.
6) To prevent large circulating currents in the ground plane from disrupting operation of the LTC3536, all small signal grounds should return directly to GND by way of a dedicated Kelvin route. This includes the ground connection for the RT pin resistor and the ground connection for the feedback network.
7) Keep the routes connecting to the high impedance, noise sensitive inputs FB and RT as short as possible to reduce noise pick-up. Example from MODE route in case the chip is synchronized with external clock.

The V3 design of switching regulation circuit is a failure, however, those results and experience will be transferred to the V4 design in the future.

3.4.2 Low Power High Performance Computer

Small Form Factor (SFF) is a general term which implies the dimension is smaller than the standard size of computer case and motherboard. The typical SFF computer is usually smaller than Micro-ATX (244mm x 244 mm), however, the most common SFF in recent times is the mini-ITX size measuring at: 170mm x 170m.

The biggest problem with the ITX SFF is the thermal management inside of a SFF case. A typical SFF case, such as Shuttle SK21G and SS59GV2 cases, has been studied in regards to the conjugated heat transfer problem in which ambient air is the final heat transfer medium in [40]. The Shuttle SK21G and SS59GV2 have cool air intake from total of 2 small side grilles and 1 small front grille. The processor generated heat has been transferred through a heat pipe system and pushed out through the back opening grille, as shown in Figure 49.

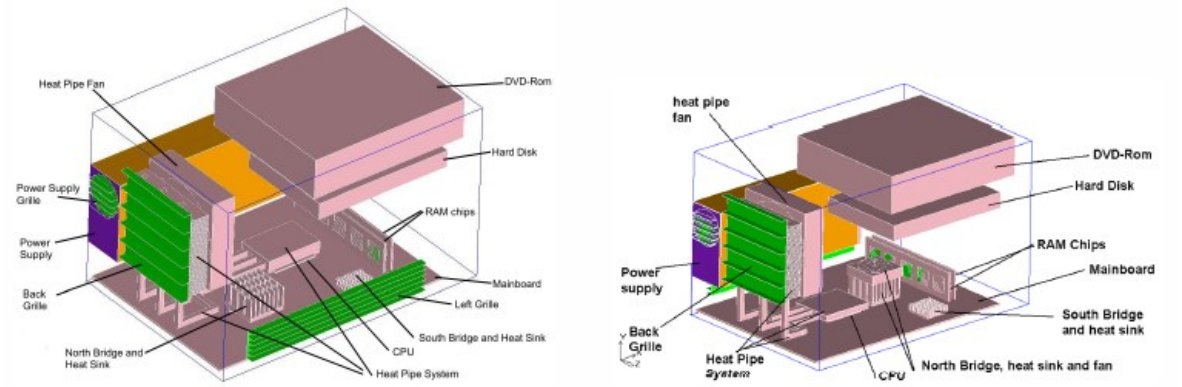


Figure 49. Shuttle SK21G and SS59GV2 in icepak model

Generally speaking, the air circulation goes from the front/side to the back, but when a video card is installed vertically next to the CPU, the heat generated by the video card stays within the case, and the only way to remove them is by the same exit as the CPU fan in the back. This works in the Shuttle case and it consists of a lot of empty space for the heat to move around.

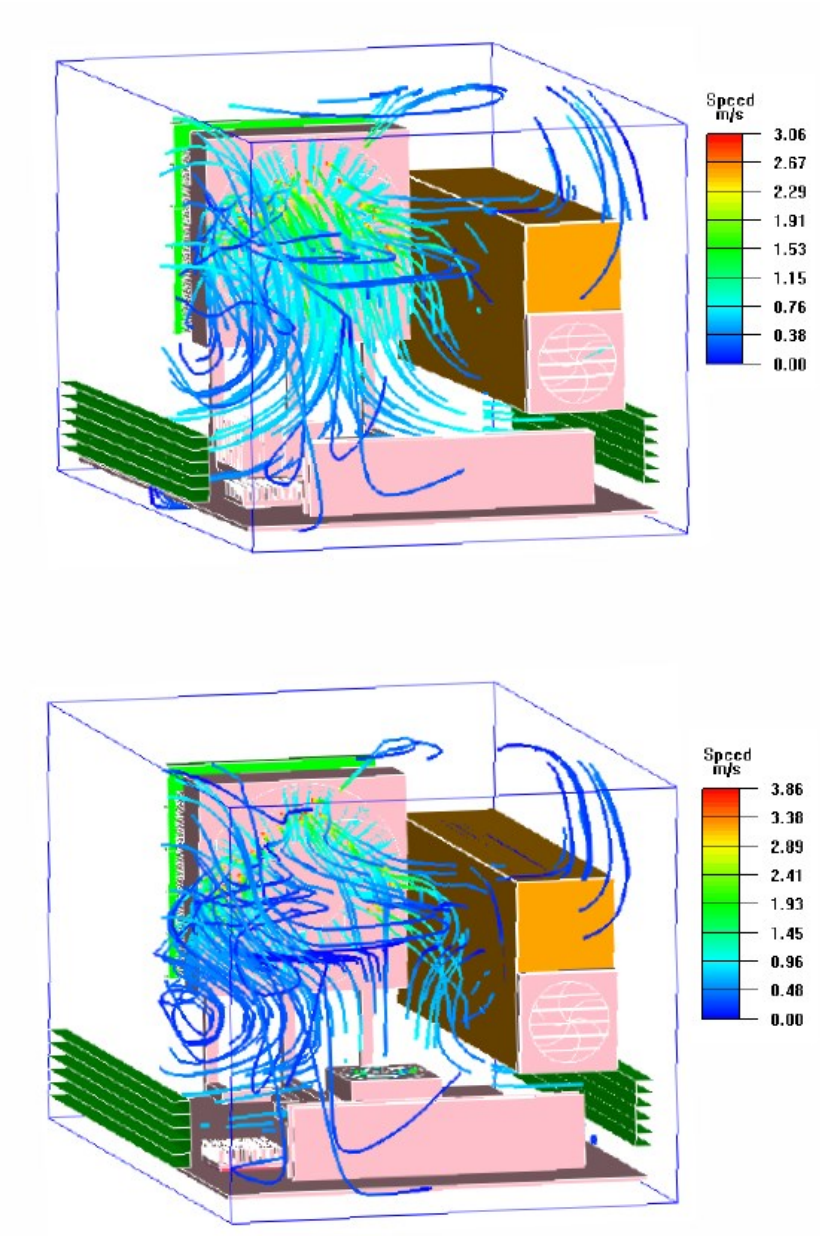


Figure 50. Particle traces for SK21G (top) and SS59GV2 (bottom) with video card installed

Shuttle ITX SFF with the dimension of 300(L) x 200(W) x 185 (H) mm is a relatively large case with lots of room to spare. Since the mobile robot tank has limited space, a smaller and much more compact case

design is needed. The more compact the case, the more important the thermal management is, especially when a high performance GPU and CPU are utilized as shown in Figure 50.

3.4.2.1 Casing Design

The goal of the HPC case design is to achieve minimum foot print with maximum cooling capability for an ITX motherboard and Nvidia CUDA GPU. This case design is capable of 250 W TDP heat dissipation.

This case is built with 5 mm thick acrylic sheet and total dimension at 195 mm x 195 mm x 125mm with air vent grill on all six sides to ensure air intake and exit.

The top layer of the case takes cool air from the side and vents out the hot air from the top vent grill, while the bottom case also takes cool air from the side but vents the hot air out from the bottom. The bottom ventilation is not ideal because hot air usually has tendency to rise, however, since the bottom layer is hosting Hard Disk Drive (HDD) or Solid State Disk (SSD), the heat ventilation is not that crucial. The designed case is shown in Figure 51.

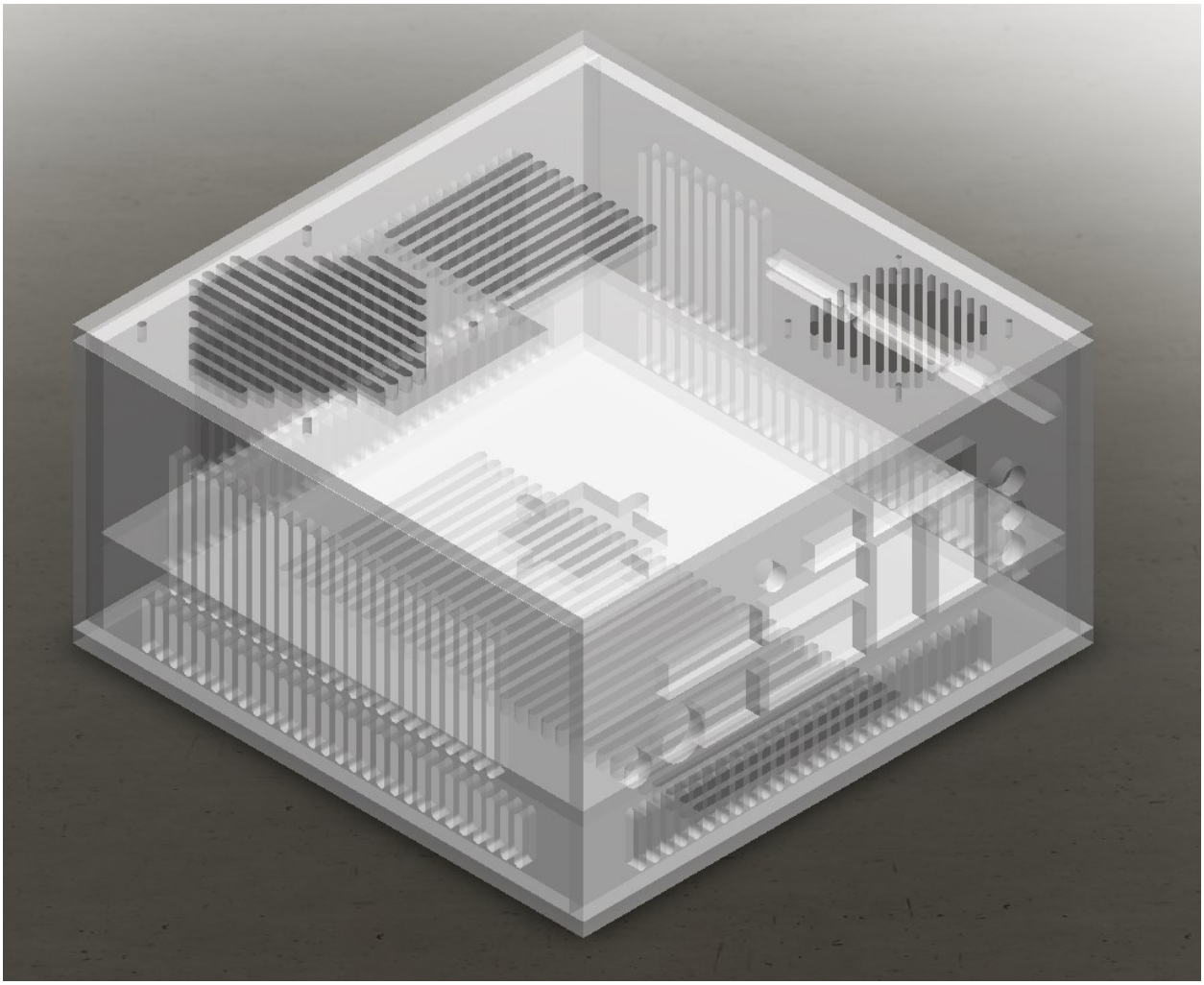


Figure 51. 3D render model of LoP HPC casing design

3.4.3 SBC System

A Samsung S3C2440 10" developer board with touch screen has been utilized and deployed on the mobile robot tank as a visual feedback. However, the default operating system is a Fedora Embedded Linux, which does not translate well with the portability of my code. To ensure ease of development, Fedora Embedded Linux must be replaced with a Windows based system.

Microsoft Windows Compact Embedded (WinCE) is an operating system vastly different than the regular Microsoft Windows. Unlike traditional windows that allow you to install the operational system first and drivers afterward, WinCE requires you to compile the system with "exact" specifications with driver source files and compile or cook them into a ROM image for deployment.

To compile a WinCE, it requires specific version of Microsoft Visual Studio. WinCE7 Tech Preview requires ARMv 5 or higher architecture; however, the Samsung S3C2440 is based on ARM920T which is an ARMv4. WinCE6 and WinCE5 have support for ARMv4.

I experimented with both WinCE5 and WinCE6 and found out that both behave similarly. However, Samsung S3C2440 requires Board Support Package (BSP) and core kernels in order to compile. A BSP package consists of various source files in C, configuration header files, boot ROM strap, reference files, VxWorks image, and other initialization files. The comparison between WinCE5 and WinCE6 is shown in **Table 11**.

Table 11. WinCE build comparison

Configuration	Windows CE 5.0		Windows CE 6.0 R3	
	Foot Print Size	Boot Time (Sec)	Foot Print Size	Boot Time (Sec)
Headless - Core	480 KB	4 Sec	512 KB	5 Sec
Headless + TCPIPv4	3.9 MB	6 Sec	4.5 MB	7 Sec
Headless + TCPIPv4 + Wifi	5 MB	9 Sec	5.1 MB	9 Sec
GUI - Core	11 MB	7 Sec	13 MB	7 Sec
GUI + TCPIPv4	13 MB	10 Sec	13 MB	10 Sec
GUI + TCPIPv4 + Wifi	15 MB	12 Sec	15.2 MB	12 Sec
GUI+COM+Media	18 MB	14 Sec	19 MB	13 Sec
GUI+COM+Media+.Net2.0	24 MB	15 Sec	25 MB	15 Sec
GUI+COM+Media+.Net2.0/3.0	28 MB	17 Sec	29 MB	15 Sec

Windows CE, like any other embedded system, allows maximum size of ROM that is 50% or less of the available RAM. This is due to the fact that once an embedded device is powered on, the exact image of the ROM will be loaded directly into RAM and while it is on, the RAM will consist of the entire operating system. This is the reason why rebooting any embedded system takes longer than expected, but the response time once it is loaded is great. The down side to it is that the device requires consistent energy to power the RAM in order to keep the operating system alive.

This is true to every modern embedded operating system, such as iOS, Android, and so on. However, there are ways to get around it by ROM slicing, which is to partition the ROM into different sections and

either to clear the parts of RAM or to load the needed module depending on the situation. This feature is not there in Windows CE 5.0 or 6.0 or even 7.0. In Windows 8 RT, ROM slicing is added.

Based on the experiments, it clearly shows that a fully working Windows CE can be as small as 480 KB and as large as half size of the RAM which is 64MB. As a result, the maximum size of the ROM is 32MB (30MB of OSes and 2MB of bootstrap). When it comes to headless operation, WinCE5.0 shows advantages on both size and boot speed, but as soon as the advanced features are added, the advantage of boot speed drastically decreases.

The difference between WinCE5.0 and WinCE6.0 is not only in kernel but also in memory management, as shown in Figure 52. The change of kernel drastically reduces the amount of calling time and system process as shown in Figure 53.

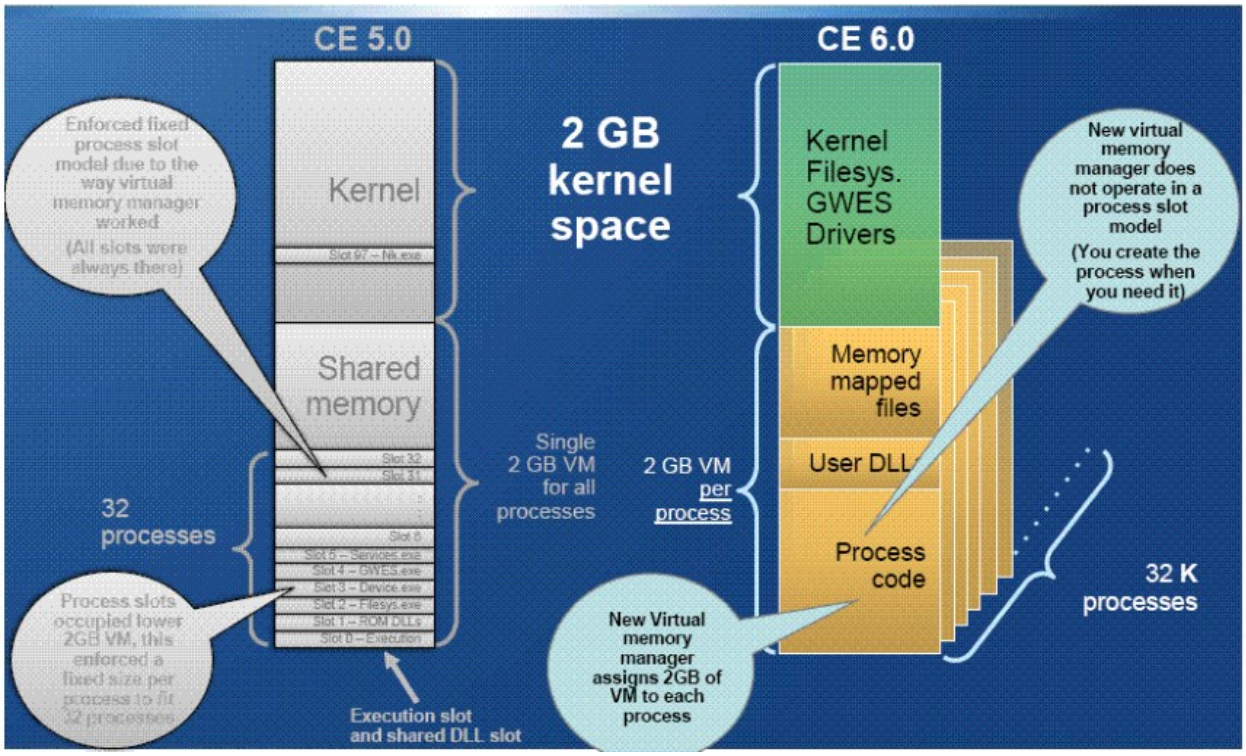


Figure 52. Windows CE 5.0 & 6.0 memory models[5]

[5] Windows Embedded CE 6.0 Advanced Memory Management (http://msdn.microsoft.com/en-us/library/bb331824.aspx)

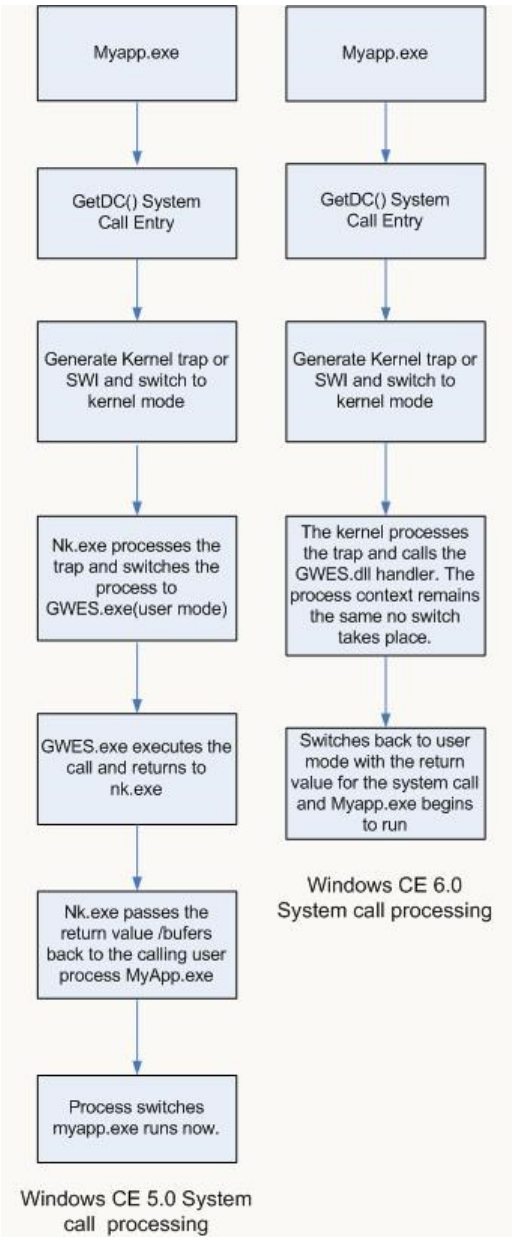


Figure 53. Windows CE5.0 & 6.0 kernel and system calling processing

At end of the day, a Windows CE 6.0 R3 ROM is created and tested with COM+ configuration and .Net Compact Framework 2.0 & 3.0 built-in for the Samsung S3C2440 system. A simple relay program is created utilizing the COM+ module and .Net Compact Framework 3.0 as a visual feedback for the mobile robot tank.

3.4.3.1 SBC Visual Feedback Program

The SBC visual feedback program communicates with the HPC through RS232 serial communication. The SBC and HPC is a one way communication as SBC only receives the information and displays back on the LCD screen in a visual Microsoft Windows form fashion.

The HPC sends 2 types of information: voice commands and logic commands. Voice commands are the commands feedback from the HPC to indicate if the voice command has been recognized, if so, what the identified commands are. Logic command is also a visual feedback that displays the decision that is made based on the computation of raw data from Microsoft Kinect for XBOX. The information received by the SBC is 8 characters long and is broken down into 2 parts as shown in Figure 54. The first part starts with a letter "V" which indicates a voice command feedback, and the 3 digits that follow indicate the type of command from the command list. The second part starts with a letter "L" indicating a logic command feedback and the 3 digits that follow also indicate the type of command from the command list, which is given in **Table 12**

V = Voice Command Feedback. L = Logic Command Feedback.
x x x = types of command from the command list.

Figure 54. SBC visual feedback information

Table 12. SBC feedback command list

Type		
Voice Command	V000	Finalizing
	V001	Initialing
	V009	Idling
	V011	VC: Robot Start
	V021	VC: Robot Backward
	V041	VC: Robot Turn Left
	V051	VC: Robot Stop
	V061	VC: Robot Turn Right
	V081	VC: Robot Forward
	V999	VC: Unrecognized Voice Command
Logic Command	L000	Finalizing
	L001	Initialing
	L009	Idling
	L011	LC: Robot Start
	L021	LC: Robot Backward
	L041	LC: Robot Turn Left
	L051	LC: Robot Stop
	L061	LC: Robot Turn Right
	L081	LC: Robot Forward
	L091	LC: Robot Stop – No Way Out!
	L999	LC: Unrecognized Logic Command

3.5 System Implementation Conclusion

Microsoft Kinect SDK provides a quick and easy method to access Microsoft Kinect sensor's raw data with Microsoft C# on the Microsoft .Net Framework. This streamline the entire development process as Microsoft has already ensure the compatibility and performance of the integration in regards to the above elements. Within the mobile robot tank controller program, Microsoft Speech Platform and SDK empower the speech recognition capability of the program, while OpenCV/Emgu compute the image with the assistance of NVidia GPGPU through NVidia CUDA/managedCUDA. Last, the mouse tracking capability is utilized SlimDX for C# implementation.

The Nvidia GPGPU is located on the HPC (High Performance Computer), which does all the image processing with help of CUDA/managedCUDA and OpenCV/Emgu. While HPC's x86 Processor does the other computations that are deemed ineffective on GPU, such as: mouse tracking, Microsoft Speech Platform…etc.

The hardware design of the mobile robot tank is powered by the V2 linear regulation circuit instead of V3 switching regulation circuit; this is due to the design flow and improper component selections for the switching regulation circuit. The low power high performance computer provides all the computational needs in a small and low power enclosure located on the robot itself while the SBC provides all the visual feedback that is needed.

Chapter 4 Tweaking and Optimization

The first version of the mobile robot tank controller application has a lot of performance problems and resource management issues.

The V1 has 90% - 100% CPU utilization on an Intel quad core system with threading. The threads are dynamically created over the course of the operation and lack proper termination once the threading operation is completed.

To improve the response time and overall efficiency, the raw data conversion process, depth processing, and CPU threading has been revisited, improved and tweaked.

4.1 Raw Data Conversion Optimization

Every depth data requires a format conversion to iImage. The initial version requires 2 processes: 1) take 16 bits depth raw data and rearrange it into iImage 2D short array; 2) Take the newly created iImage 2D short array and down sample it into a 2D byte array. This process creates latency up to 35ms per frame or up to 1000ms for 30 frames. This process consists of two read and two write operations from and to memory.

The first optimization of this process is to combine the 2 processes into 1: first, take the raw depth image and bit shift it to remove the 3 masking bits; next, apply down-sampling to the same raw depth data array; finally, rearrange the 1D short array into an iImage 2D byte array. This optimization allows an average of 10ms latency reduction per frame and an average of 500ms for 30 frames.

This latency was contributed by two factors: the conversion processes that use additional float variables as a buffer and the rearrangement process that employed the division/reminder method which increases the computation latency.

To ensure smoother conversion, two loops have taken place of the division/reminder method and the removal of the excess float buffer variable. As a result, the latency of 30 frames per second is improved to 200ms.

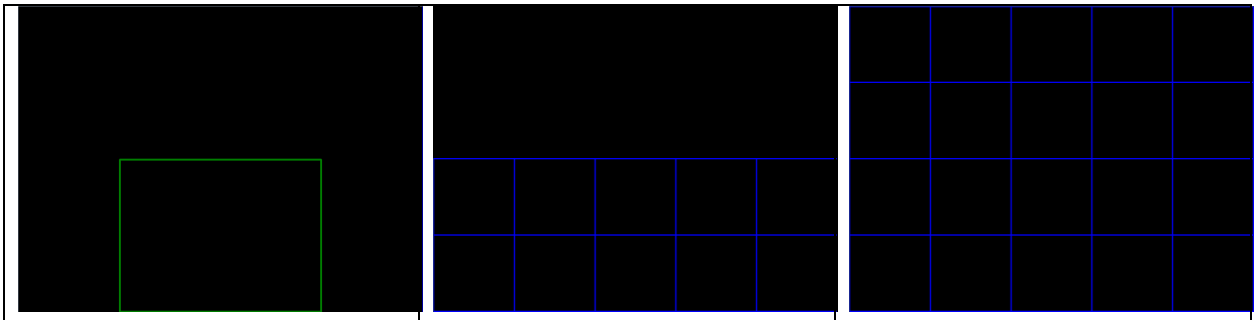
4.2 Depth Processing Revisit and Tweaking

The obstacle detection in V1 processes only the lower half of the image frame and computes only “one” ROI. It does not allow resizing of the ROI or testing the ROI by different method. The V1 also has high resource consumption when more than 3 objects are highlighted. The performance degradation exponentially increases as more objects are highlighted. [41]

The other problem is scalability with multicore processing, which has to do with the types of threading instruction sets and the method of threading implemented for the depth processing algorithm.

All of those above lead to V2 development. A complete new multithreaded design has been proposed for depth processing with a custom-built collision engine. The previous design only utilized one fixed ROI area and is not scalable or modifiable in any way. The initial design for V2 was a 5x2 ROI region grids.

However, the effectiveness of threading allows me to scale to 5x4 ROI regions without any performance impact, as shown in Figure 55



V1 ROI Region (1x1)	V2 Initial ROI Region (5x2)	V2 Final ROI Region (5x4)

Figure 55. Comparison of different ROI designs

In the V2, in order to provide scalability in the case of finer detection detail by increasing amount of ROI regions, each ROI region is an independent object and the number of the ROI regions can be increased without limitation. There are a few problems: 1) the ROI detection algorithm in V1 does not work with multiple ROI regions. 2) The highlighted obstacles or objects are not uniformly shaped and testing the boundary against multiple ROI regions will lead to high computation consumptions. 3) The same Graphic User Interface (GUI) debugging implementation from V1 is not efficient for displaying multiple highlighted objects. 4) Remote debugging or visual feedback needs to be done real-time with minimum data transfer.

To tackle problem 1), a complete new detection design is used, which consists of CUDA implementation and threaded CPU implementation for fall back. In the CUDA implementation, the image frame is divided in factors of 48 pieces for each CUDA core with each piece composed of 80x80 pixels and is scalable when more CUDA cores are available. Each of the 80x80 pixel piece is processed by Gaussian smooth filter and is gone through the contour processing.

Once an object is identified, it is sent to a share memory with the object size, coordinates and counts. Then when all pixel pieces are completed, a new process takes a look at the locations and the size of contour as a whole picture, eliminates any overlapping objects, and sends this final matrix back to the CPU.

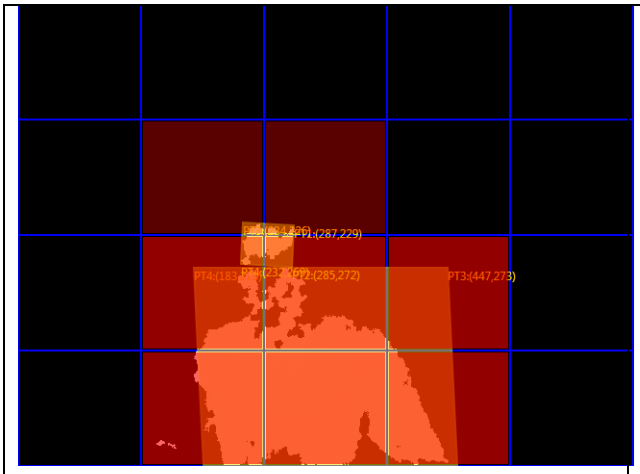
The CPU implementation is a little bit different. It utilized separable Gaussian smooth matrix first, and then applied contour function from OpenCV to find the object. Once an object is found, a rectangular box will be generated to cover the entire area of the found object. At the same time, in the background, the individual ROI region will be prepared for collision test.

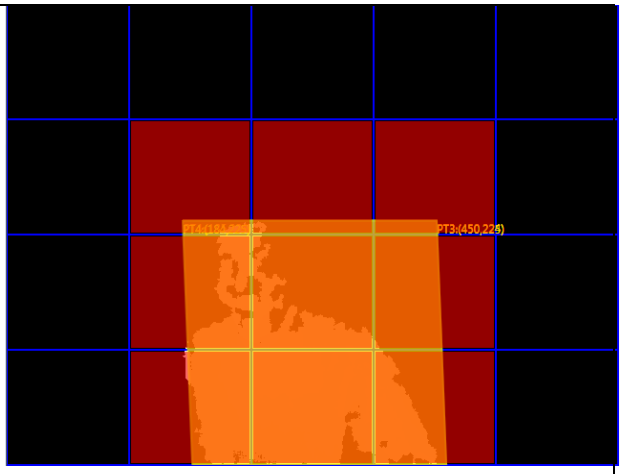
To solve problem 2), an effective and scalable collision engine is needed for computing the location of object. To this end, each ROI region is created as an object with fixed size and location, and all ROI regions are sent to a thread pool for collision test.

Each detected object is enclosed by a rectangular box, so that the collision detection can be made easier. Each small blue grid in Figure 56 is spawned as individual threads in the thread pool, which allows faster collision detection in regards to various detected blobs.

Since the collision detection is only processing the edge of the rectangular box for the detected blob, there are high possibility that the box covers multiple ROIs and leaves the middle of the surrounded ROI undetected. In this case, a checker table is employed. The checker table is a fast Boolean table which is triggered by the corresponding element in the ROIs to the true state. Otherwise it will remain false. Once all the ROIs have triggered the checker table, another function will initialize and look into the table to checker table by filling the triggered element gaps. This step is to ensure that all the proper ROIs are triggered properly by the rectangular box.

The other functionality of ROI collision detection process is to allow immediate detection of the change in the detected blob, such as: resizing, merging, separating and tracking. Each detected blob rectangular box object has finite amount of threads that keeps track of the properties of the detected blob object. Hence, each object has its own designated threads and when the object is no longer insight or intrusion by the nearby object, the program will process the next frame in order to decide if the collided object is still single object or multiple object or the possibility of missing frame.





“2” Separated Objects Detection	“2” Separated Object Merged into “1”

Figure 56: Collision engine: object merging and separation

4.3 CPU Threading

Multithreading processors have been around since the 1950s and the conceptualization of Simultaneous Multithreading (SMT) [42]was only introduced in 1968 in an IBM research lab. SMT is a hardware implementation of multithreading to enhance the overall efficiency of superscalar CPUs. [43]

The first commercial SMT implementation was the Alpha 21464, which was developed in 2001 by DEC with collaboration with Dean Tullsen of the University of California, San Diego, and also Susan Eggers and Hank Levy of University of Washington. However, the Alpha processor was discontinued shorty due to the acquisition of DEC by HP and in favor of Intel Itanium CPUs. Dean Tullsen's work on SMT implementation was soon implemented in Intel's CPUs.

Intel first introduced Intel Hyper Threading Technology (HTT) on Intel Pentium 4 and Intel Xeon in 2002. Intel HTT design allows every physical processor core to address two logic cores or virtual cores and share the workload between them when resources are available. In order for this technology to work, modern operating system (OSes) support is needed. In some operatingl systems, the virtual cores may appear as two processors and the OS has to schedule the processes and tasks between the virtual cores at once. In addition, two or more processes can share the same resource. In the case of process/task failure, the resource can be freed up for the next task in queue, as shown in Figure 57.

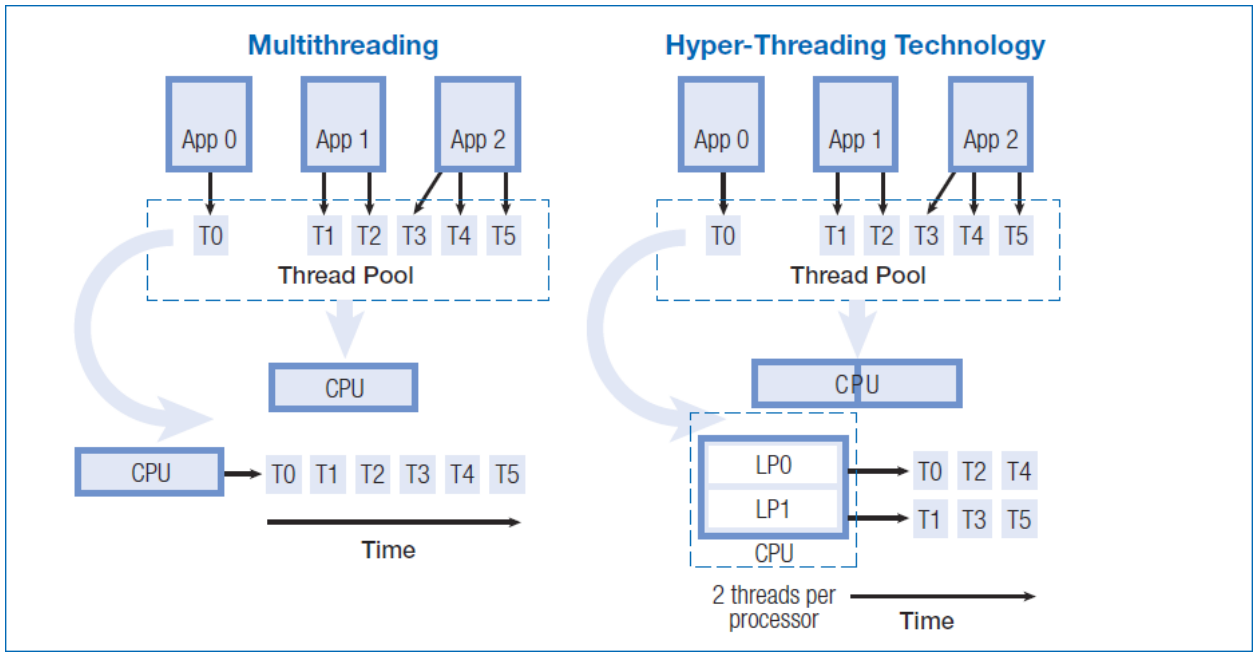


Figure 57. Intel's HTT scheme [44]

Modern application design employs both sequential and parallel tasks. Multithreading exploits the parallelism in the application work load by executing the parallelizable tasks in parallel. The application performance is improved by the degree of parallelizable tasks and available resource for executing.

The Intel HTT allows speed up by 50% per physical processor core. However, the projected performance gain is only possible with proper implementation of code modules, functions, instructions and dependencies.

Amdahl's law [45] summarizes the theoretical maximum speedup based on the workload, as shown in Figure 58.

$$\mathrm{T_{sequential}} = \text{ total time to complete work } (1 - \mathrm{P}) + \mathrm{P}$$

$$\text{Performance Improve } = \frac{\mathrm{T_{sequential}}}{\mathrm{T_{parallel}}} = (\frac{1}{1-P} + \frac{P}{N} + O)$$

where P is the number of parallelizable tasks, O is the overhead of multithreading and N is the number of the physical processors.

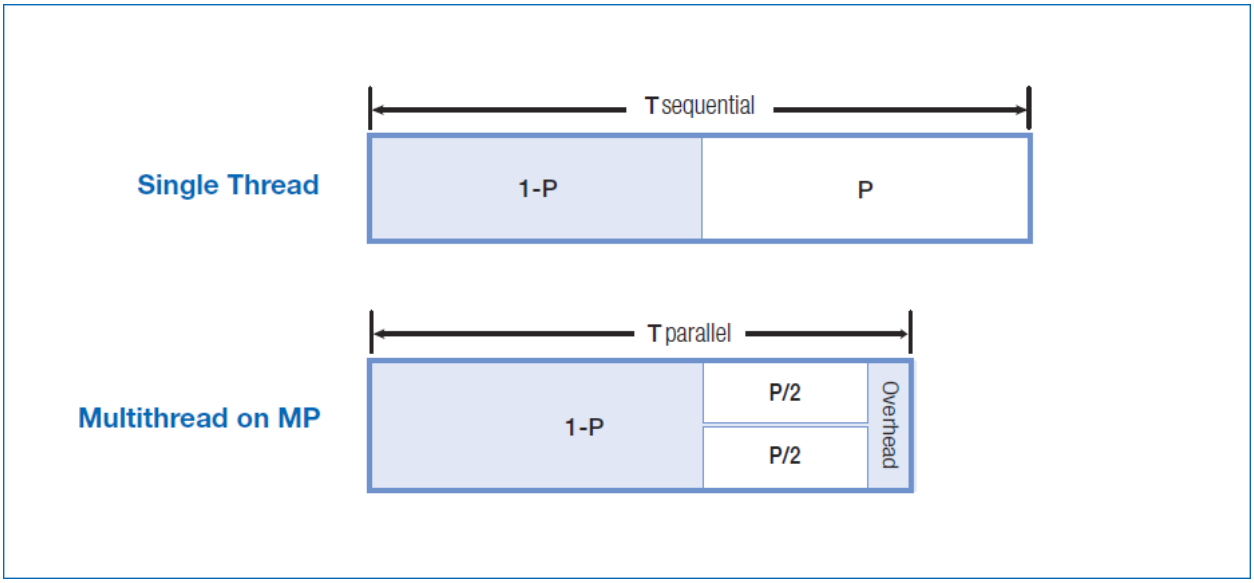


Figure 58. Amdahl's law in MPs

Theoretical performance with utilization of Intel hyper threading has a 50% improvement; however, due to the overhead, a single physical core processor can only gain a maximum of 33% performance boost.

Microsoft C# 3.5 or higher provides all 3 types: traditional threading, background worker, and thread pool. Each of the threading methods has its own advantage and disadvantage. In the mobile robot tank controller, all 3 types are utilized and so is Intel TBB (Thread Building Block).

The traditional thread has a high startup overhead and typically takes up at least 1MB per pre-allocated thread. Ideally traditional threading is pre-created at the initial stage of the program. Dynamic initialization of traditional threading spells troubles, as traditional threading does not allow creating of additional treads within the created thread. In order to create more threads in the course of time, the main program/thread has to create, control, and clean them up. This forces the main threads to be a monitor and manager of the entire traditional threads.

However, the advantage of traditional threads allows you to limit the amount of threads created and has full control over them. In the mobile robot tank controller, 3 additional traditional threads are created during the initial stage of the program for: raw data acquisition, audio processing, and image processing. All of these 3 threads remain alive throughout the course of the program until the program terminates.

Background worker, like the name implies, is a background thread that is primarily utilized in GUI responsiveness and updates when needed. Background worker is an event-driven thread and it spends most of the time idling until it is triggered by an event. However, the implementation of a background worker is an asynchronous process, which means it is a non-blocking process, in which this process does not need to wait for completion or dependency of other process.

Two background worker threads are used for GUI notification. One thread is designed for updating logic commands: graphical feedback and logic decision. The other thread is used for audio processing updates: notification of commands, confident level and results. There are few design concerns which lead to the utilization of the background worker implementation in the mobile robot tank controller. 1) The GUI should be used only when the CPU is free because it is not essential. 2) The GUI front end can be turned on or off. The background worker turns out to be the best fit because it requires the least amount of resources, it can be spawned dynamically, and it can be triggered when it is needed or is completely discarded.

The last type of threading is the threading pool which is introduced with .Net framework 3.5. This method of threading is the least amount of manageability due to the first-in-first-out design: if there is a queue of 100 tasks in a thread pool, then, it will use as many threads as have been created to service these requests (usually a much smaller number than the total tasks, 10 in this case). However, the numbers of the threads are limited by hardware and driver, but this can be overcome by utilizing the hardware provider API such as Intel TBB. The thread pool will consistently re-evaluate and re-inquire the status of existing threads in 500ms intervals. If the evaluation has determined that these tasks are short and quick, it will keep reusing the same thread without initializing additional threads. But if the workload is consistently requesting a large number of threads or a long processing time, the thread pool will create additional threads to process the request.

The thread pool is resource aware and resource conscious, and sounds like a programmer's dream, but the awareness only occurs after multiple runs of the exact "same code" and similar outcomes in order for the thread pool to learn the behavior and workload of the code. This implementation is often seen in mobile device implementation, where the most common application will be given more resources and a more rapid response. In Windows, the first implementation is the "ReadyBoot" with Intel "Turbo Memory" in Microsoft Windows Vista.

The hardware/software implementation bundle is attempting to speed up boot time and response time by analyzing the commonly used software and behavior of the code. This allows the system to preload the most commonly used software into a NAND flash memory space. However, due to the diversity of

the programs installed in a PC and limited hardware NAND memory space in Intel Turbo Memory, it was "never widely adapted". [46]

In the mobile robot tank controller, thread pool is utilized within the traditional threading. The raw data acquisition has its own thread pool to pull the different raw data off Microsoft Kinect sensor. In audio processing, a thread pool is implemented for grammar builder/library to ensure full parallelism for all the recognizable commands. The image processing has also its own thread pool for all the filters that are implemented. Figure 59

Table 13: Benchmark of different C# threading methods

Thread Types	Initialization Time	Memory Overhead	Switching Time	Termination Time	Execution Time
Traditional Threads	50ms	3MB	>500 ms	100 ms	155 ms
Background Worker	<5 ms	<1MB	>100 ms	< 5ms	> 500ms
Thread Pool	15ms	1 MB	100 ms	50 ms	300 ms

```
static void Main(string[] args)
{
   TraditionalThreadCreation();
   Console.Read();
}

static void TraditionalThreadCreation()
{
    // this guy will do even numbers
    thread1...
    // this guy will do odd numbers
    thread2...
}

static void Task(object p)
{
    for (int i = int.Parse(p.ToString()); i <=10000000; i += 2)
    Console.WriteLine(i);
}
```

Figure 59. Code snippet – thread benchmark code

4.4 Tweaking and Optimization Conclusion

Raw data conversion optimization provides a quick glance on how a simple optimization can impact the whole program. This simple optimization allows 5 times performance improvement and this simple change allows 30 frames to be process within less than 200 ms instead of original 1000 ms.

The depth processing revisit and tweaking allows better scalability and improvement in object detection, this processing in conjunction with CPU threading allows real-time processing without dropping frame or putting the entire system to a halt. The overall improvement after the tweaking and optimization is huge; from the previous 13 frames per second, to current 30 frames per second.

Chapter 5 Future Work and Conclusion

Throughout the course of designing and implementing of the mobile robot tank controller there are many design considerations found along the process and many of the bugs are fixed to ensure stability. However, since this is still an early stage of development, the main focus of future work will be based on scalability, portability and stability.

5.1 Scalability and Portability

In the mobile robot tank controller's design, the object detection is fully threaded and implemented with scalable algorithms on both CPU and GPU; however, the primary coding techniques and APIs are utilizing Intel x86 architecture on an x86 Microsoft Windows.

In order to achieve portability, some of the code needs to be ARM assembly compatible so that they can be run in ARM architecture. Since the code are built on top of Windows .Net Framework, there are a few foundation functions which are required to be modified for smooth transition to Compact .Net Framework. One of the examples is the "Canvas" and "Pen" functions, as both of them do not exist in Compact .Net Framework, the direct substitutions are "Ink Canvas" and "Solid Brush" functions instead.

The mobile robot tank controller utilizes Intel Threading Building Block (Intel TBB). If the CPU architecture is changed to ARM, the threading techniques need to match the generation of ARM architecture. For example, both NVidia Tegra 2 SoC and Samsung Exynos 4210 are ARM dual-core Cortex A9 but NVidia Tegra 2 is missing NEON threading extension.

Due to various implementations of ARM SoC, even with the same architecture/generation, some of the features may be missing. This leads to specific coding for the selected SoC.

5.2 Problems, Bugs, and Issues

Microsoft Kinect sensors utilize 60-65% of a single USB 2.0 controller as shown in Figure 60. A typical computer consists of 2 USB 2.0 controllers and as many as 12 USB ports share across the 2 controllers. Generally speaking, a typical x86 or ARM computer can only host up to 2 Microsoft Kinect sensors or 3 if the system does provide an additional USB 3.0 controller; however, the USB 3.0 controller will be limited to 480 Mbits instead of USB 3.0 5Gbits/10Gbits. This is due to the Microsoft Kinect sensor running at USB 2.0 specification, hence, the USB 3.0 controller has to downgrade to 2.0 to ensure compatibility.

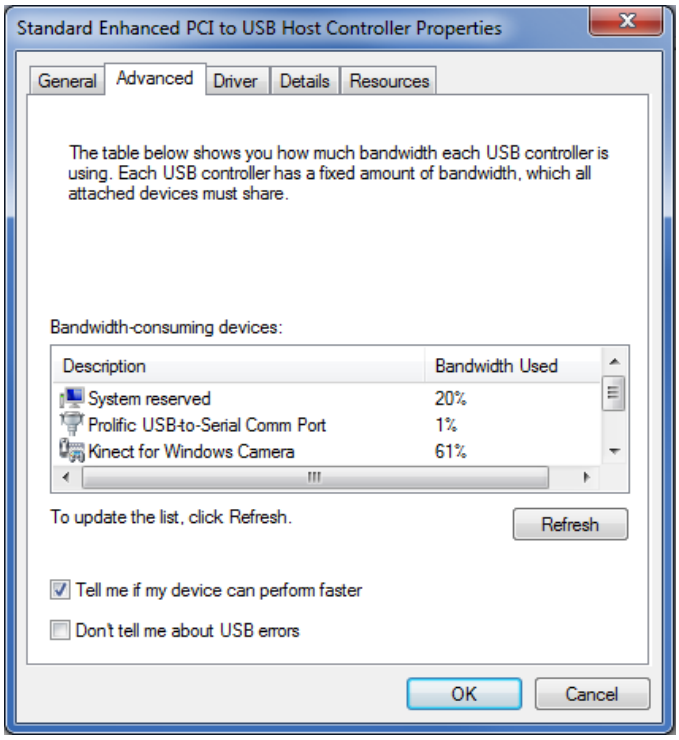


Figure 60: Microsoft Kinect sensor USB bandwidth occupancy

It is interesting to note that not all enhanced USB (USB 2.0) controllers are created equally and operating systems play a major role of monitoring and distributing the available USB bandwidth.

The mobile robot tank utilizes only 1 Microsoft Kinect sensor at the moment. During any initial run of the software, both software and hardware will behave according to the design. However, once the program quits and restarts, the system indicated the following error: “insufficient USB bandwidth” as shown in Figure 61. One solution is to unplug the Microsoft Kinect sensor and connect it to another enhanced USB controller, but, once the program restarted the current and previous enhanced USB controller still indicate that: “insufficient USB bandwidth”. To solve the problem completely, the system has to be shut down and restarted.

Figure 61: Insufficient USB bandwidth error messege from Microsoft Kinect status

The phenomenon occurs only if the onboard HPC system and backup HPC are mounted on the same motherboard. However, I was unable to duplicate the same error with another computer. I am speculating it might have something to do with the lack of proper grounding on the mobile robot tank, specific hardware issues and operating system issue with scheduling or driver.

There is a possible fix by building an USB software reset and garbage collect feature into the mobile robot tank controller software, which might be able to release the occupied USB bandwidth by Microsoft Kinect sensor.

5.3 Thesis Conclusion

When it comes to image processing on GPU, there have been tremendous computational time saving advantages over CPU, however, the complexity of memory sharing management and limited program accessible memory storage increase the difficulty in program design in comparison to the traditional CPU programing.

There are a few cases where GPU computation has no significant time saving over CPU programming, which is extremely true in sequential programing or recursive computation where the previous result is needed in order for the next computation. For example, audio signal processing requires probability of the first word in order to calculate the possible outcome of the second word and the final result of the sentences is based on each word. In this case, GPU computation has no advantage at all.

In V2 ROI computation, the raw data is divided into 20 pieces and each piece is computed by a fixed but scalable number of cores in GPU, and as a result, one frame is being processed by multiple cores in parallel at the same. The processing time of a frame drastically decreases and up to 10x speed up per frame base on 96 CUDA cores. With increases of CUDA GPU cores, the speed up would also increase.

The overall construction and design of this mobile robot tank indicates the importance of Control Engineering and Electrical and Computing Engineering. The logic in this mobile robot tank controller program is extremely simplistic, but in order to achieve complete autonomy and to have more

interactive capability, mathematical logic models such as predictive controller and complex decision design such as neural network should be implemented to increase capability and efficiency.

This entire master thesis of robot design has proven the importance of GPU computation, the proper threading and resource management in both CPU and GPU, and also the criticalness of the tweaking and optimization during the data manipulation and processing. The preparation and outcome planning has been deemed necessary throughout the courses of the design process. If an outcome was not pre-determined, additional code will be needed to deal with that specific scenario based on that outcome; however, this additional code also limits the scalability and expendability of the program, even worse, creating complications or errors in other possible scenarios.

In summary, this mobile robot tank with V2 mobile robot tank controller software is capable of following a “STOP” sign while attempting to avoid obstacles and voice recognition with predetermined voice command library with acceptable timing.

Appendix A - Pseudo Code

DSP:

(Data Size)

FromDSP = 23 with CRC, 21 without CRC ToDSP = 16 with CRC, 14 without CRC

- Waiting for Data (ToDSP)
    1. Data Start with “U”
    2. Data Start with “L”
    3. Data Start with “R”
    4. If all of those data are valid
- Send Data to PC (FromDSP)
    - Data Start with “C”
- In any time if data received has “S” at beginning, then DSP starts the timer and runs DSP program
- In any time if data receive has “P” at beginning, then DSP stops all motors and reset the entire process

HPC:

(Global)

- Initialize Kinect sensor Hardware
- Initialize global variable storage
- Initialize the following classes (Speech, Depth, Color, logic, Serial, and HID tracking)
- Initialize events handlers
- Initialize GUI

(Speech Recognition)

- Initialize corresponded language library
- Initialize local audio variables
- Initialize 4 seconds audio buffer
- If incoming sound > threshold noise level
    1. recode the audio, and stop as soon as audio signal drops below threshold
    2. process audio buffer with speech engine
    3. match word with library
    4. return highest possible matching words to main program
    5. execute corresponding audio command
- start again

(Depth Data Acquisition)

- Check if Kinect is ready, if not quit
- Grab depth data from Kinect
    1. Bit shifting depth data to remove player info
    2. Convert shifted depth data to iImage
    3. Scale depth data if necessary
    4. Apply different image processing to the current iImage frame on GPU
    5. Return the processed iImage back to main program
- Process this iImage
- Start again with next frame

(Color Data Acquisition)

- Check if Kinect is ready, if not quit
- Grab color data from Kinect
    1. Convert color data to iImage
    2. Conduct Color filtering / noise reduction and smoothing if necessary
    3. Apply different image processing to the current iImage frame on GPU
    4. Return the processed iImage back to main program
- Process this iImage
- Start again with next frame

(Logic)

- Check processed iImage data from color and depth for validation. If it is good, proceed; if not discard it and wait for next frame
- Process results obtained from both depth and color iImage
- Pass decision to corresponded events trigger and trigger robot movement through serial communication class

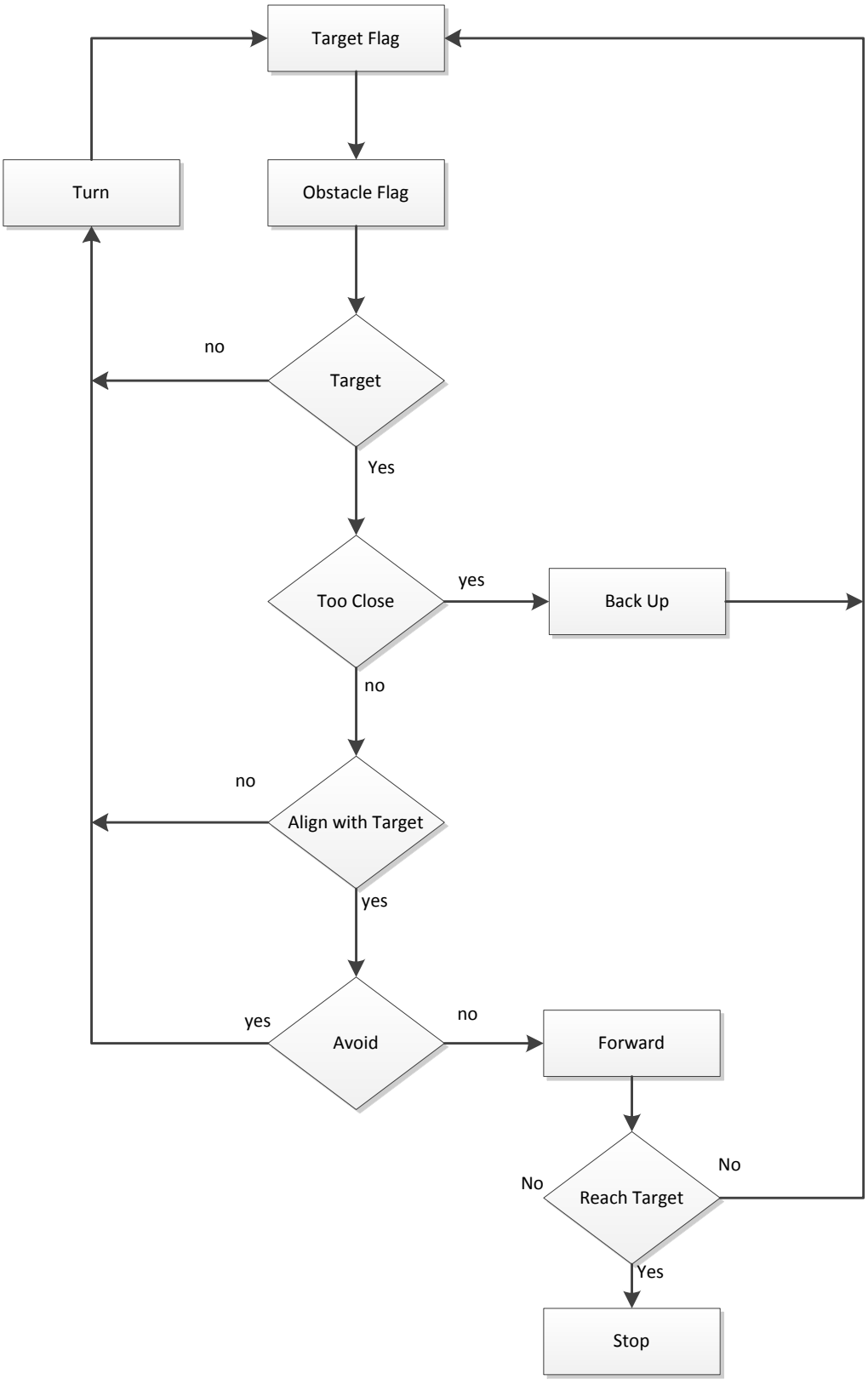
(HID tracking)

- Check for valid existing HID with built-in *.ini file. If valid, proceed; if not, request for new HID ID.
- If robot starts in first time, reset the HID location to (0,0); if not, keep it as is
- Any movement change will be traced by HID class
    - Calculating turning/moving angles and distance

(Serial Communication)

- Initialize buffer base on CRC or non-CRC setting
- Initialize sending and receiving buffer
- Send following data to DSP
    - Data Start with “U”
    - Data Start with “L”
    - Data Start with “R”
- If above data is valid, receive the following
    - Data Start with “C” to confirm continuing sending movement information
    - Decode incoming data and translate to proper PWM and motors ID
    - Send the rearranged data with “R” to DSP for motor movement.

Appendix B – V3 Logic Flow Chart



References

[1] Encyclopedia of Science and Technology, McGraw-Hill, 2009, pp. 594-596.

[2] M. G. Miran Gosta, "Accomplishments and Challenges of Computer Stereo Vision," *52nd International Symposium ELMAR-2010,* pp. 15-17, September 2010.

[3] Z. Zhang and T. Kanade, ""Determining the Epipolar Geometry and its," *International Journal of Computer Vision,* vol. 27, pp. 161-195, 1998.

[4] M. Gosta and M. Grgic, "Accomplishments and Challenges of Computer Stereo Vision," Croatian Post and Electronic Communications Agency, University of Zagreb, Faculty of Electrical Engineering and Computing.

[5] M. Bleyer, *Segmentation-based Stereo and Motion with Occlusions,* Vienna University of Technology: Ph.D. dissertation,, 2006.

[6] C. Loop and Z. Zhang, "Computing Rectifying Homographies for Stereo," in *IEEE Computer Society Conference on Computer Vision and*, 1999.

[7] R. Szeliski, *Computer Vision: Algorithms and Applications,* 2010: Springer.

[8] L. Lamel, F. Lefevre, J.-L. Gauvain and G. Adda, *Portability Issues for Speech Recognition Technologies,* Orsay: Spoken Language Processing Group, CNRS-LIMSI, 91403.

[9] M. Aynehband, A. Rahmani and S. Setayeshi, "COAST: Context-aware pervasive speech recognition system," in *Wireless and Pervasive Computing (ISWPC),*, Hong Kong, 23-25 Feb. 2011.

[10] eFunda, Ine. Efunda.com, "Rapid Prototyping: An Overview".

[11] R. Marron, "C# Implementation of SLAM Using the Microsoft Kinect," 2012.

[12] A. Das, "Process Time Comparison between GPU and CPU," Hamburger Sternwarte, , 2011.

[13] Intel, "Intel® Core™ i7-960 Processor Ark Infomation portal".

[14] Nvidia, "Nvidia Geforce GTX 280 Spec".

[15] W. V. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal and P. Dubey, "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU," IEEE-ACM, Saint-Malo, 2010.

[16] A. Shinsel, "Implementing Laser Scanning for Robotics Vision with CUDA," 2010.

[17] I. R. Victor Adrian Prisacariu, "fastHOG - a real-time GPU implementation of HOG," Oxford, 2009.

[18] Wikipedia, "Infrared".

[19] ipac, "Near, Mid & Far Inferared".

[20] *LR4 Laser Rangefinder Interface Data Sheet,* 2013.

[21] R. Müller-Albrecht, "Optimized for the Intel® Atom™ Processor with Intel's Compiler," 2010.

[22] Intel, "Intel Atom Processor N200 Series Performance DataSheet," 2011.

[23] Intel, "Intel Atom Processor D500 Series performance datasheet," 2011.

[24] Intel, "Intel Core 2 Duo Processor T9000 Series performance datasheet," 2011.

[25] Intel, "Intel Core 2 Quad Processor Q9000 Series for Mobile," 2011.

[26] Intel, "Intel® Core i5-500 Mobile Processor Series," 09/08/2011.

[27] Intel, "Intel Core i3-300 Mobile Processor Series Performance Datasheet," 2011.

[28] Intel, "Intel Core i5-500 Mobile Processor Series Performance Datasheet," 2011.

[29] Intel, "Intel Core i7-600 Mobile Processor Series Performance Datasheet," 2011.

[30] PASSMARK Software, "CPU Benchmark".

[31] VIA Technologies Inc., "VIA Nano Processor Introductory White Papaer," 2008.

[32] S. J. Gardner, "Low-Power Processors: When Performance MAtters (VIA Nano X2 comapred to Intel Atom)," 2011.

[33] Nvidia, "Geforce 8800 & Nvidia CUDA - A new Architecture for Computing on the GPU".

[34] NVidia, "NVIDIA's Next Generation: Kepler GK110," 2012.

[35] Microsoft MSDN, "Kinect for Windows Architecture".

[36] M. Rozak, "Talk to Your Computer and Have It Answer Back with the Microsoft Speech API," *Microsoft System Journal,* January, 1996.

[37] A. Oliver, B. C. Wünsche, S. Kang and B. MacDonald, "Using the Kinect as a Navigation Sensor for Mobile," ACM, Auckland, 2012.

[38] Jaycar Electronics, "DC-DC CONVERTERS: A PRIMER," 2001.

[39] Linear Technology, "LTC3536 Datasheet," 2011.

[40] I. Tari and O. E. Orhan, "Numerical Investication on Cooling of Small Form Factor Computer Cases," *Engineering Applications of Computational Fluid Mechanics,* vol. 2, no. 4, pp. 427-435, 2008.

[41] D. S. O. Correa, D. F. Sciotti, M. G. Prado, D. O. Sales, D. F. Wolf and F. S. Osorio, "Mobile Robots Navigation in Indoor Environments Using Kinect Sensor," in *Second Brazilian Conference on Critical Embedded Systems*, São Carlos, SP – Brazil, 2012.

[42] Wikipedia, "Simultaneous multithreading".

[43] M. Smotherman, "End of IBM ACS Project," 25 May 2011.

[44] Intel, "Intel® Hyper-Threading Technology Technical User's Guide," 2003.

[45] G. Amdahl, ""Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," in *AFIPS Conference Proceedings (30)*, 1967.

[46] B. Crothers, "Intel 'Braidwood' chip targets snappier software," 2009.