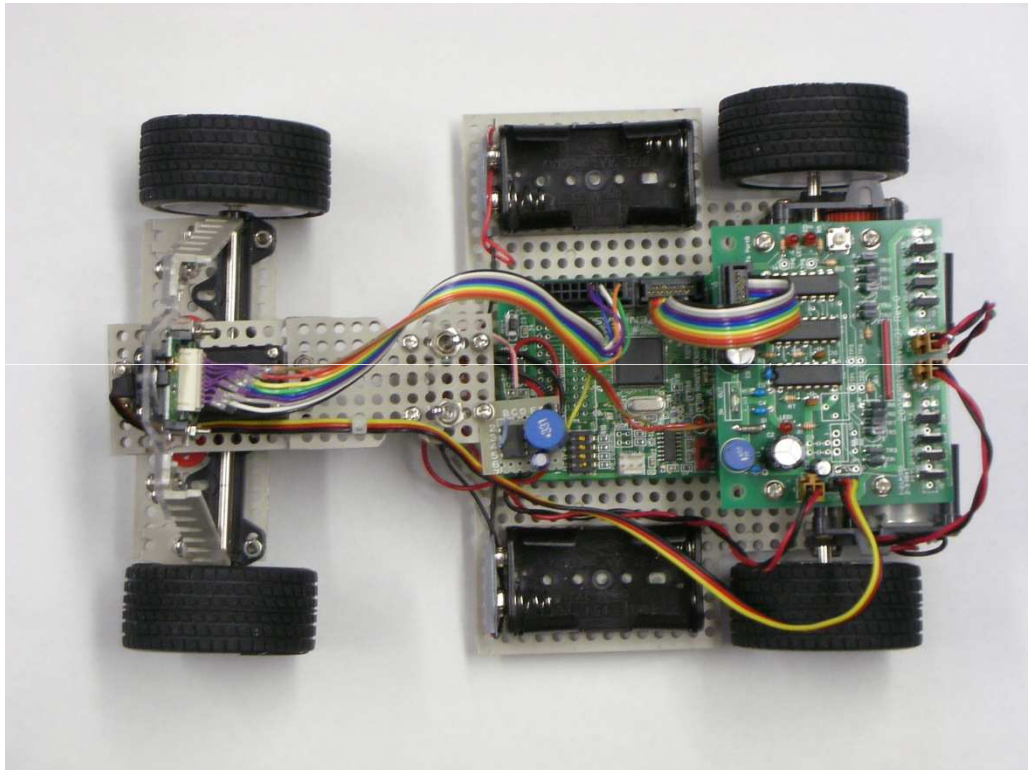
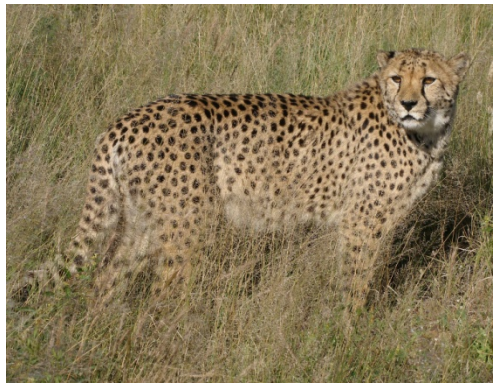
画像処理で走るマイコン･カー



EyeChan2008
溝上洋三氏(X線技術研究所)

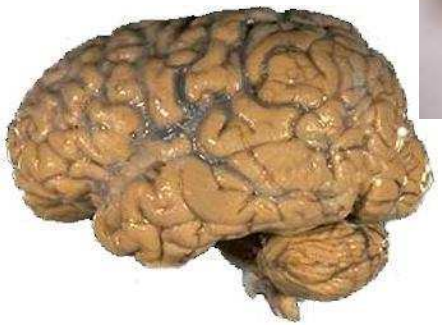
1．今、私たちは、歴史上どの過程にいるのか。

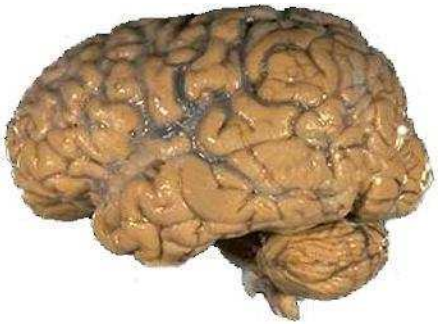
人類の歴史と情報との関係



人間より、早く走る動物、空を飛ぶ事ができる、鳥など、運動能力では,人間より、優れた動物は、沢山存在します。
しかし、人類は、現在地球上での、食物連鎖の頂点に立っています。
なぜでしょうか。

人類の最も大きな特徴は、大きな大脳を持っていると言う事です。





- 脳の大きさだけが、全てではなく、その入出力機能として、言葉を使える事。
- 声帯が備わっている事が、高度な情報の処理能力を可能にしています。

人類の情報処理における、進歩過程。
言葉の発明

言葉は、物（認識できる）と、単語を１対１にする（ここでは、話の都合上単純化して考える）事です。

- **犬の場合。**
  - 隣の田中さんを見て、「ワン」。
  - 次に、佐藤さんを見て、「ワン」。
  - 横田さんを見て、「ワンワン」
  - 福田さんを見て「ワン」
  - 小林さんをみて、「ウーワン」。
- **人間の場合。**
  - 隣の田中さんを見て、「田中さん」。
  - 次に、佐藤さんを見て、「佐藤さん」。
  - 横田さんを見て、「横田さん」
  - 福田さんを見て「福田さん」
  - 小林さんをみて、「小林さん」

この犬が９９人の人間を知っていて、３種類の言葉（ワン、ウーワン、ワンワン）を喋ると仮定します。

情報量を計算してみる。

犬の場合

- ９９人知っていて、その他の人は知らないので、９９＋１＝１００通りの認識が可能ですので、１００の2を低とする対数をとれば、約6．65です。
- 即ち、認識対象物の情報量は約6．65Bitです。
- 同じように、犬の言葉の情報量を計算すると、約１．58Bitです。
- すなわち、言葉の情報量が少なくて、犬の言葉では、情報の欠落が発生します。

人間の場合はどうでしょうか。
•仮に全ての、認識物にたいする、言葉を1対１で、持っていたとします。
•即ち、前の人の名前の例で行くと、認識物6．65Bitに対して、言葉も6．65Bitありますから、情報の欠落なしに、伝える事ができます。
•以上が、人類が情報出力機能としての、言葉の意味であります。

情報を受け取る。

- もちろん、情報伝達ですから、耳は、その言葉を聴いて、大脳で同数の種類に認識する
- 聞いた人が、話し手と共通に認識できる人が半分の４９人場合は、５．６５Bitしか伝わらず、1ビット欠落します。

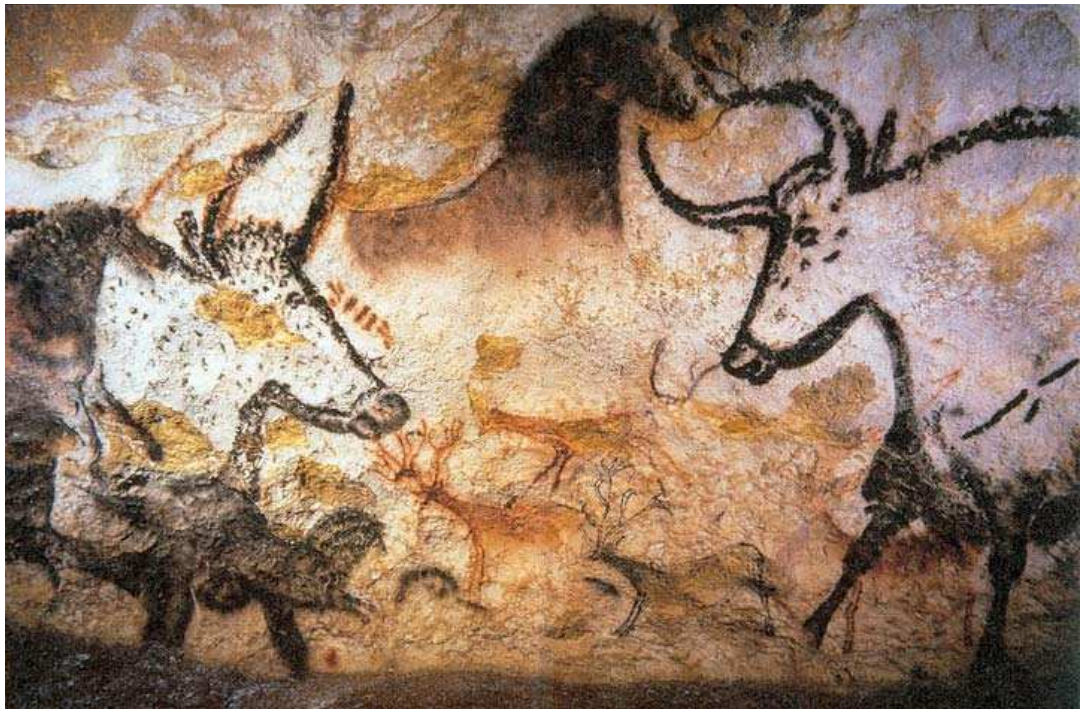
言葉は、情報処理に制約を与える

- 言葉は、出力機能はシリアライズ（RS-232Cみたいに）された、ものすので、人類の情報処理に大きな制約を与えました。
- 即ち、私たちの理論展開が、知らず、知らず、時系列（フローチャート）の発想になってしまいます。

－－　後ほど、情報処理の方法について、考える事にしましょう　－－

絵の発明

- 人類が言葉を使うようになったのと、絵を描くようになったのは、どちらが、先でしょうか。（随分昔の話で、人類はその時の事を、本して残していないでしょう。）
- 言葉の情報伝達手段と、絵による、情報伝達では、何処がどうちがうのでしょうか。



言葉の情報伝達手段と、絵による、情報伝達では、何処がどう違うのでしょうか。

- 言葉による物とは違い、その一瞬写す写真と同様に、時系列が存在しません。（時系列を表現するには、絵本とか、絵巻物の様に、複数の絵を、連ねる方法があります。）
- 絵は、一瞬に多量な情報を伝える事ができます。

  例えば、ラジオの様な簡単な回路でさえ、回路図なしで、言葉にだけで、話し手と、同じイメージを、聞き手が頭の中に、書き出す事は困難でしょう。
- 音声は一瞬で、消えますが、絵は、時間が経過しても、情報を保存できます。

文字の発明。

- 絵を単純化して、言葉の様に、「物」と「単純化した絵」を1対１で対応さすと、言葉の特徴である、情報の単純化と時系列で論理を表現できる。
- しかも、絵の特徴である、時間が経過しても、情報を保存できと、言う機能が付加されます

その結果。

- 情報を、世代を超えて、正確に残す事ができる様になりました。
- しかし、この世の物の数だけ、「単純化した絵」を作るらないと、1対１に対応させれないので単純化した絵を作って、その絵と、物との対応を覚えるには、数が多過ぎます。

しかし

- この世の物の数だけ、「単純化した絵」を作らないと、1対１に対応させれないので単純化した絵を作って、その絵と、物との対応を覚えるには、数が多過ぎます。

表意文字と表音文字

- 発音と絵を1対１に対応する事ができれば、絵の種類は、数十種類で表現可能となります。
- したがって、漢字の様な、表意文字から、アルファベットの様な、表音文字へ進化しました。
- しかし、パソコン時代の現在では、アイコンという表意文字の復活があります。

紙の発明

- 文字を、石、粘土板等に書いていた、時代は、多量の情報を持ち運ぶには、不可能ですが、紙の発明により、情報の持ち運びが可能になりました。

印刷機の発明。

- １４４７年のグーデンベルクによる、活版印刷の発明により、多種類、多量の本を作る事が可能になりました。
- 活版印刷の発明以前は、本の量が少なく、その本を読める人は、貴族、神官などの特別な地位の人で、人数がかぎられていました。
- しかし、本が多量に発行されると、庶民も、本を読む事が、可能になりました。

その結果。

- 活版印刷の発明以前の本から、知識を得る事ができる、特別な地位の人が、地球上に１０万人いたと、仮定します。
- 活版印刷の発明により、本から知識を得る事ができる人が、１億人になったとします。
- すると、今まで、１０００年かかった、文明の発達が、1年で可能になります。
- 活版印刷の発明以後、人類は、急速に文明を発達させ、現代の科学文明を実現させました。

近代の発明品（1）

- 写真の発明。

  「百聞は、一見にしかず。」の言葉どうり、画像が沢山の情報を含んでいます。

- レコードの発明。

  従来、音声は、一瞬で消えますが、録音する事で、音声の、この弱点を克服できました。

- 電信器の発明。

  電信器の発明は、情報の即時性を可能にしました。

  江戸と浪速の間を、書類で、情報伝達する事を、考えてみると、分かるとおり、書類では、離れた所との、通信に即時性がありません。

- 電話の発明。

  電信器は、文字をコード化、して送る為に、誰でも、簡単に、使用する事はできません。

  しかし、電話は、だれでも、簡単に使用できます。

近代の発明品（２）

- FAXの発明。

  多量に情報を含んでいる、画像を送る事ができます。

- 無線機の発明。

  無線機発明以前は、有線では、点と点の間の通信しかできません、でしたが、無線機の発明により、地球上、何処でも、通信可能になりました。

- テレビジョンの発明。

  1926年に高柳博士によって、世界で初めての、テレビが作られた。

コンピュータの発明。

- 真空管式で動作したのは、１９３９年のアタナソフ＆ベリー・コンピュータが最初。
- コンピュータとしては１９４６年のENIACが最初と言われています。
- 言葉の発明に始まる、情報の歴史は、①情報の表現方法、②伝える（空間的、時間的）方法の改良でした。
- それは、情報を処理するのは、人間が前提の話です。
- しかし、コンピュータの発明で、情報処理をするのは、機械でも可能になりました。
- 従って、人間の情報処理能力（速さ、多量の情報量）の限界を超える事が可能になりました。

情報の発見

- １９４８年シャノンが、情報理論を発表。
- ユートン力学的な考えでは、質量の無いもの（空間を占めない物）は、この世に実在しない物と科学者は、考えていた。
- シャノン以前は、例えば、質量のない精神の様な物は、科学者の扱い範囲外であったが、シャノンの登場により、情報と言う質量のない物が、科学の分野で扱える様になりました。

これからの発見、発明。

- 電子技術の進歩と、情報理論の登場により、コンピュータは、急速な進歩を遂げています。
- 人工知能の実現。

  高度な知能は物を正しく認識する必要があります。

  したがって、人工知能の実現には、言葉の処理より、画像処理が是非とも必要です。

  言葉の処理から、知能を得よう言う、研究が世界中で行われて、います。しかし、言葉は、人間のフィルターで得た、情報を、人間の感性で、表現したものなので、人工知能の実現には、既に、その知能体が、人間の感性を持っている事が、必要です。

  人間の感性は、内部に矛盾があっても、良いのですが、理論が大変です。
- 自我のあるコンピュータの登場。
- 物思いにふけり、何も出力しない、電気だけ消費する、コンピュータの発明。
- 等々、色々、多数。

コンピュータのプログラムの歴史。

- コンピュータに動作を人間が司令（プログラム）する方法について、考えてみましょう。
- 「言葉は、情報処理に制約を与える」の項目で述べた様に、私たちの理論展開が、知らず、知らず、時系列（フローチャート）の発想になってしまいます。

機械コード入力方法。

- 最初のコンピュータは、決まった順序で、数値計算をする、計算機でした。
- 従って、①数字を読む、②4則演算をする、③計算結果の数字を出力する、機能が必要です。
- 動作が簡単なので、データとプログラムコードを紙テープ、又はカード等への、穿孔で可能でした。
- 従って、全て、人手による、紙テープ、スイッチでの機械コード入力です。

アセンブラの登場。

- コンピュータが少し複雑になり、プログラムコードの種類が増えると、人間に理解しやすい、文字でプログラムを記述する必要が生じてきました。
- 例えば

  １１０００００１１　００００００００　００１００００１０とコードすると、

  Z80CPUでは、プログラムカウンターに0x0022を入れろと言う命令になりますが、非常にわかり難く、沢山の命令コードを覚えるのが、困難でまた、間違いの発見もできません。

  そこで、JMP　0x0022　と書くと、人間にもなんとなく、理解が可能になります。
- 従って、アッセンブラは、機械コードとニーモニック（命令）が1対１で対応します。

FORTRANの登場。

- 数値計算する場合、数式を記述する必要があります。
- 例えば

　２００＋３００＊５＋４０／（５＋３）の計算をコンピュータにさせたい場合はどしょうか。

　ALU（数値計算部）の入力がレジスタ（REGn）とアキュムレータ（ACC)が可能として、演算結果は、全て、アキュムレータに出力されると仮定して、アッセンブラで記述すると、

```
MOV  REG1, 300
MOV  REG2, 5
MLT  REG1, REG2
MOV  REG3, ACC
```

ーーーー次ページに続くーーーー

```
MOV  REG1, 5
MOV  REG2, 3
ADD  REG1, REG2
MOV  REG2, ACC
MOV  REG1, 40
DEV  REG1, REG2
ADD  ACC,  REG3
MOV  REG1, 200
ADD  ACC,  REG1
```

で答えはACCに求められます。

- フォートランは、形式変換（formula　translation）の意味で、最初に書いた数式を、そのまま、コンピュータが解釈して、機械語に翻訳します。
- 数式の翻訳方法は、逆ポーランド記法を通常は使います。
- 逆ポーランド記法は、１＋２を「１と2を足す」の様にモンゴル言語圏内の様に訳します。
- BASIC言語はFORTRANの学習用に1行毎に翻訳する、インタープリタとして開発された言語です。
- FORTRANの様な言語を、アッセブラ等と区別する為に、高級言語と言います。

COBOLの登場

- FORTRANは数値計算が目的でしたが、COBOLは、アメリカ軍が、ベトナム戦争をする為に、莫大な量の資材の管理をする為（戦争では多量の物資が横流しされる）に考えられたのが、原型と言われています。
- 機能としては、ファイルに書く、ファイルから読むが、中心になります。
- 一般の会社に於いても、お金の管理、商品の管理に便利だったので、コンピュータが会社に、普及して行きました。

ALGOLの登場。

- FORTRAN、COBOLは、非構造化言語と呼ばれます。（FORTRAN77になって構造化された）
- ALGOLはアルゴリズムの研究用に考えられた構造化言語で、C言語等の構造化言語の源流です。
- PascalはALGOLの学習用として考えられた、構造化言語です。
- 構造化言語とは

  構造化言語以前は、プログラムはフローチャートで記述するのが、普通でした。

  しかし、構造化言語は、フローチャートの基本機能である、GOTOを使わなでバグの少ない、プログラムを書ける、方法です。

  記述方法は、フローチャートに代り、ナッシ・シュナイダーマン・チャート（NS)チャートを用いると、理解しやすい。

フローチャートの考え方は、先の言葉の発明の所で述べた様に、時系列で、理論を説明する、人間の理論構成には、良くマッチする方法ですが、理論が入り組んできて、前に説明した所から、現在の位置に、どう言う理論で、たどり着いたかを、過去に戻って、行くのが、難しいのです。

それは、どの位置からでも、飛んで行ける、GOTO文があるからです。

構造化言語は、①順次、②選択、③繰り返し、の機能だけで、記述します。（GOTO文は必要在りません。）

C言語の登場。

- C言語は、ATTがコンピュータの販売を禁止されていた時に、UNIXの開発で使用された言語です。（当時は、OSはアセンブラでないと、書けないと言うのが、常識だった。）
- 従って、高級言語で構造化言語でもありますが、ここの命令の実装は、アッセンブラ・マクロの応用をした様な、柔軟性があります。
- アセンブラのニーモニックをインラインでC言語の中に埋め込む事もできます。
- ポインターの機能により、直接アドレス指定でメモリを読み書きが、できます。

PROLOGの登場

- データ間の関係をパターンマッチングできる言語です。
- 余談ですが

  日本で、第５世代（人口知能）コンピュータの開発が国家プロジェクトとして、１９８２年から１０年間に５７０億円かけて、行われた。

  　そのときの採用した言語が、既に開発されていた、PROLOG言語でした。

オブジェクト言語の登場

- コンピュータの発達で、表示画面が文字中心から、Windowsの様なGUI環境になりました。
- 従来の様に、データと処理を、別の所で管理すると、矛盾の解決に手間取る事が、増えます。
- 表示画面には、アイコン、ボタンなどの、オブジェクトが配置されますが、そのボタンが、実際のボタンをコンピュータの中に写し変えた物と考え、物（オブジェクト）は、特性（データ）と機能（プログラム）で、できていると考えて、Classの中に両方入れる事により、オブジェクトの考えのプログラムができます。

「シグマ計画」の余談。

- 日本で1985年から250億円をかけた、「シグマ計画」と言う、国家プロジェクトが、ありました。
- 内容は、情報化社会の発展に伴い、ソフトウエア技術者が不足する。
- その解決策として、日本の中央に巨大なコンピュータを設置して、個々のプログラマーは、自分の開発した、ソフトを、その巨大なコンピュータに、登録します。
- 皆がソフトを、共有する事により、同じような、サブルーチンを皆が。開発する不合理を解決できると、考えたのでした。
- これで、情報化社会では、日本は世界の先端を走れる予定でした。
- しかし、その当時は、オブジェクトの考えが無く、データと処理は別管理でしたので、他所から持ってきた、サブルーチンは閉じていません。
- したがって、他人が作ったサブルーチンの処理内容まで理解しないと、使用できません。
- 今考えると当たり前の事ですが、国家プロジェクトでした。

情報を処理しよう。

- 情報処理とは、なんでしょうか。
- 情報を処理する場合の状況を考えて見ましょう。

①入力された情報はあるのに、何も出力されない場合。

この処理系は、意味がありません。

何故なら、欲しいのは、出力される情報ですから。

②入力された情報がないのに、情報が、出力される場合。

この処理系は、出力は、情報が含まれていません。

何故なら、入力情報が無いので、無い情報は情報演算できません。

③入力された情報があり、処理を行い、出力した場合。

この場合は、情報処理として、意味を持ちます。

エントロピーは減少し（不要は情報は捨てられる）人間に理解し易くなっているはず。（暗号処理ではエントロピーは増加する？）

- この事から導き出されることは、情報処理に於いては、情報を処理するので、情報が主体で、処理は従であるべきです。
- 従来、プログラムの学習には、処理が中心で、データ構造の話が少ない様に思います。

情報の処理方法

- コンピュータは、１～数クロックが処理単位になっています。

  （パイプライン等の技術で1クロックで数命令実行できるコンピュータもあります。）
- 早く、情報を処理するには。

  ①処理単位当たりのクロック数が少ない事。（処理に4クロックかかるCPUより1クロックで処理できるCPUが速い）

  ②クロックが早いこと。（20MHzより2GHzのCPUが速い）

  ③処理単位当たりの処理量が多い事。（8Bitより32BitのCPUが速い）
- 現実に採用されている、情報処理の高速化は

  ①の方法は、DSPで採用されていて、１クロックで複数の処理を同時に行います。

  ②の考察から、CPUの様に高速クロックで、高速処理をする方法。

  ③の考察から、人間の脳が採用している様に、同時に沢山の細胞で情報処理を行う方法が考えられる。

  この方法はFPGA（Field Programmable Gate Array）と言う素子が採用している方法でもあります。

- 現在の所、全て満足した、素子はなく、用途別に最適な物を使用します。
- 高速動作が必要な、動画の圧縮、伸張は、FPGAか専用IC。
- 音声の圧縮、伸張程度の情報処理量であれば、DSP。
- CPUは、ワンチップのPICの様な制御目的のCPUから、パソコンの中のCPUまで、色々な、速度のCPUがあります。

FPGA（Field Programmable Gate Array）

- プログラム可能な論理素子（IC）は各社、呼び名が統一されていません。
- 基本的にCPLD（PLD）は積項を和演算する方法で行い、基本技術はROMです。
- FPGAはルックアップテーブル（１６Bit程度の小さなRAM）が基本素子となっていて、RAM技術の応用です。
- どちらの素子もHDL（Hardware Description Language）でプログラムします。
  一般的に、HDLは「VHDL」か「**Verilog」が使用されます。**
- どちらの言語も基本的考えは一緒です。
- 文法は、電気回路なので、入力端子、出力端子の定義をして、論理部分は、C言語の様に、IFで分岐、Forでループ、などです。
- 翻訳結果が回路になりますので、電気を入れると、全ての回路が同時に動作するので、CPUのプログラムの様に、上のコードが先に実行されると言う、事はありません。

カメラの仕様
M64282FP

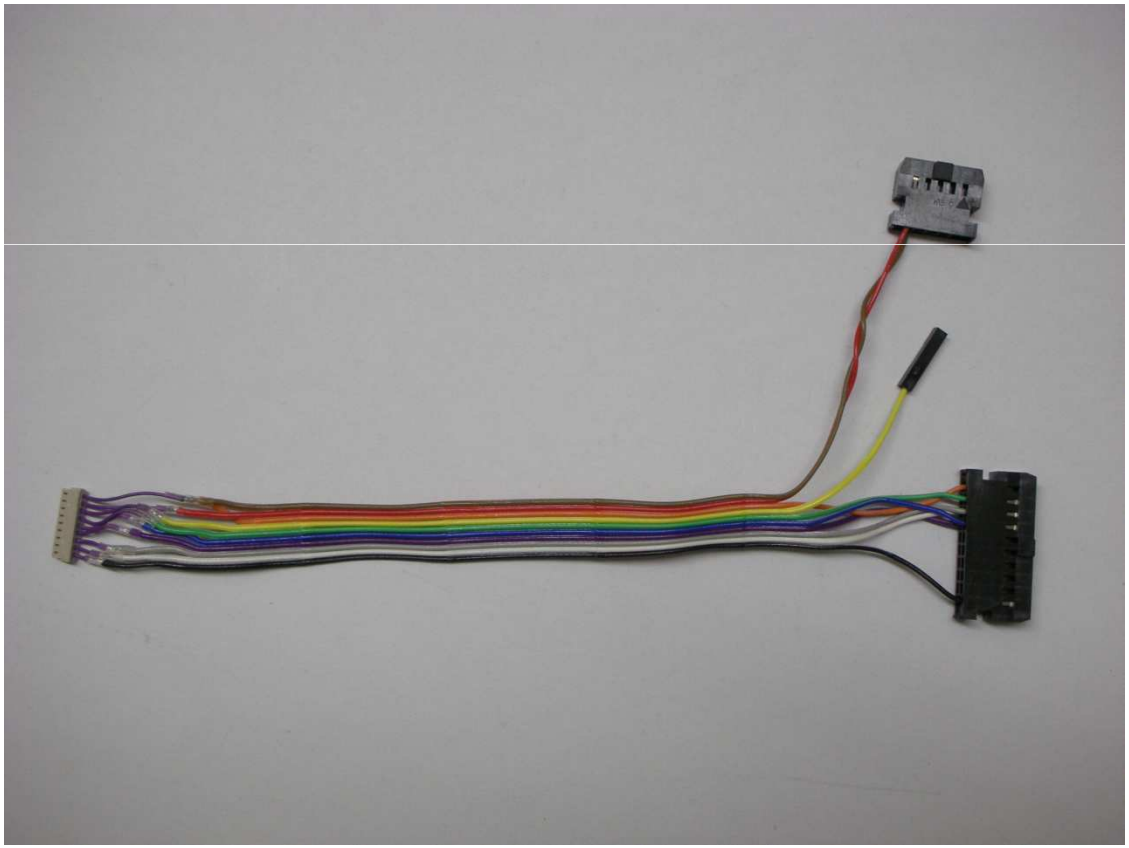
概要・特徴

M64282FP_datasheet.pdf

- 画像演算処理機能とアナログ調整機能を内蔵している。
- 128X128画素のCMOSカメラです。
- 人工網膜がもつ、情報圧縮を保有しています。
- 5V単一電源で動作
- 低消費電力
- 高フレームレート（Max16．4ms／フレーム）
- ポジネガ画像出力切り替え機能。
- 輪郭強調機能
- 輪郭抽出画像出力機能
- ゲイン・レベル調整機能
- 光学サイズ　1／4インチフォーマット
- 画像サイズ3．07X3．07mm　画素サイズ　24μmX24μm。

カメラをCPUに接続する。

- カメラへの配線は、工夫してみて下さい。



カメラ接続表

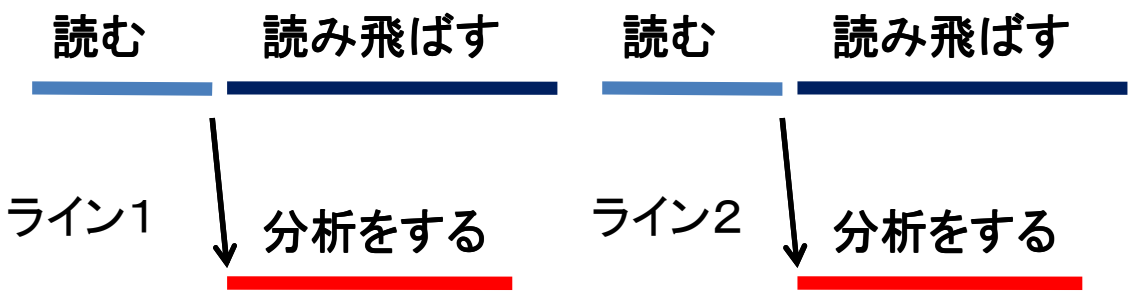
信号名	カメラのCN	ケーブルNo	CPUのCN	備考
GND	1	1	J1-10	
Vout	2	2	J1-9	AN0以外でもA/D端子で有れば可能。
GND	1	3	J6-20	
READ	3	4	J8-3	割り込み機能を使わないでソフトを組めます。
XCK	4	5	J6-17	Sio0のCLK端子に接続します。
XRST RESET	5	6	J6-13	
LOAD	6	7	J6-14	
SIN	7	8	J6-15	
START	8	9	J6-16	
VDD	9	10	J6-1	

ソフトの説明。

- カメラから画像を読む為の特別なハードウエアは積んでいません。
- 画像はソフトでクロックを出して、カメラからの出力電圧をA／D変換して読みます。（1つクロックを送ると、次の画素の明るさが電圧出力されます。）
- H8の性能を考えて

  8ライン／画面（のどの位置の8ラインはソフトで指定する）

  127画素／ラインのデータを使用する。
- 画面上のデータを読み飛ばしている時に、ソフトは、以前読んだデータを処理する必要から、読み飛ばし時のクロックは、ハード（同期通信のクロック）で行う。



主な関数

- 露光時間の調整。　ExposureSet（）
- 明るさのヒストグラム作成。　HistMake（）
- ヒストグラムから白線を検出する。HistWhitLevel（）
- １画面の画像を読み込む。　PhotoGet（）　分析処理あり。
- １画面の画像を読み込む。　NullPhoto（）　分析処理なし。
- １ラインの画像を読み込む。　M6Save（）
- ２値化処理を行う。　Nichika（）
- カメラのクロック・スタート。　CameraClockStart（）
- ソフトクロック。　SoftClock（）

関数

- カメラの初期化　　CameraInit（）
- カメラにリセットパルスを出す。 M6Reset（）
- カメラにコッマンドをセットする。Sh11（）
- DMAからの割り込み処理。　　Dma0A_int()

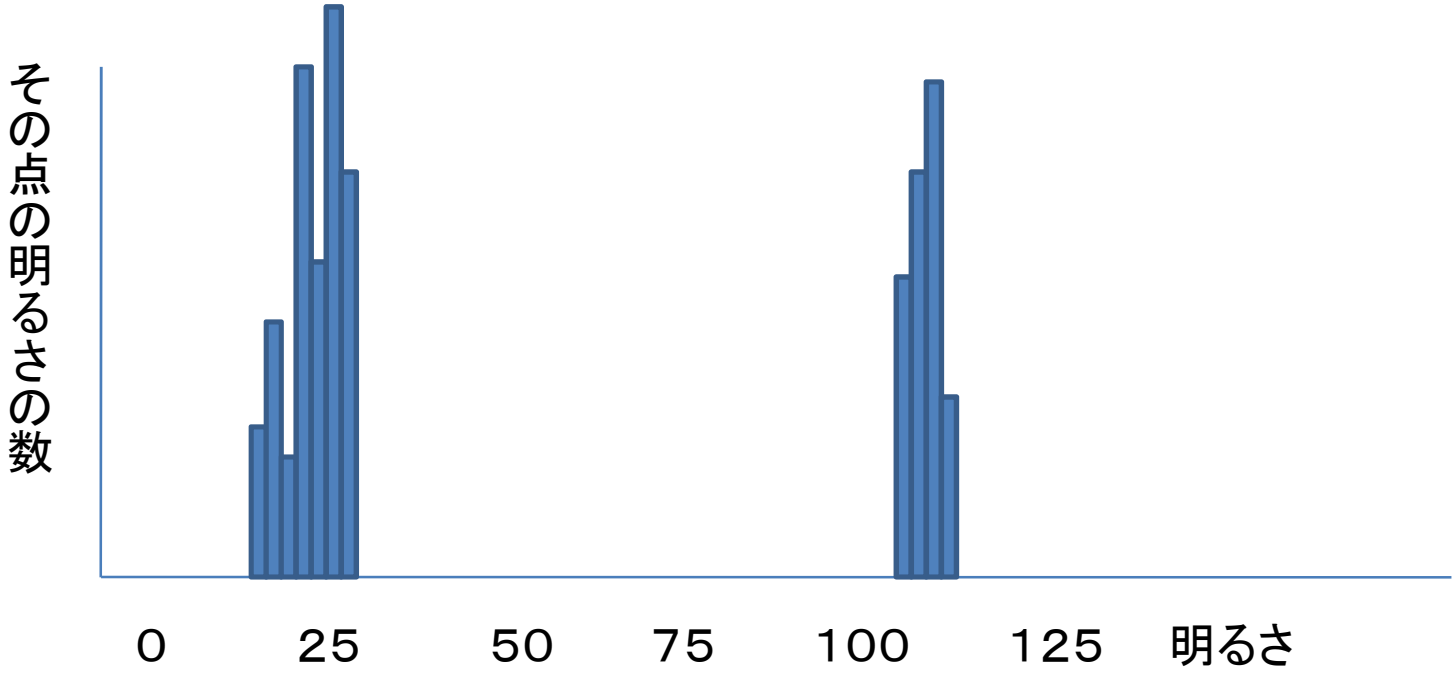
露光時間の自動調整する関数

- Uint ExposureSet(int MaxBrightV, int Offset, int Gain)
- 第１引数…白線部分の最大明るさを指定します。
  小さい値にすると、撮影（露光時間）時間は短くなります。
- 第2引数…カメラに設定するオフセットです。
  （カメラのレジスター設定を参照。）
- 第3引数…カメラに設定するゲインです。
  （カメラのレジスター設定を参照。）

ヒストグラムを作成する関数

どの程度の明るさの点（画像は点の集まり）が有ったか表にする。

- void HistMake(void)

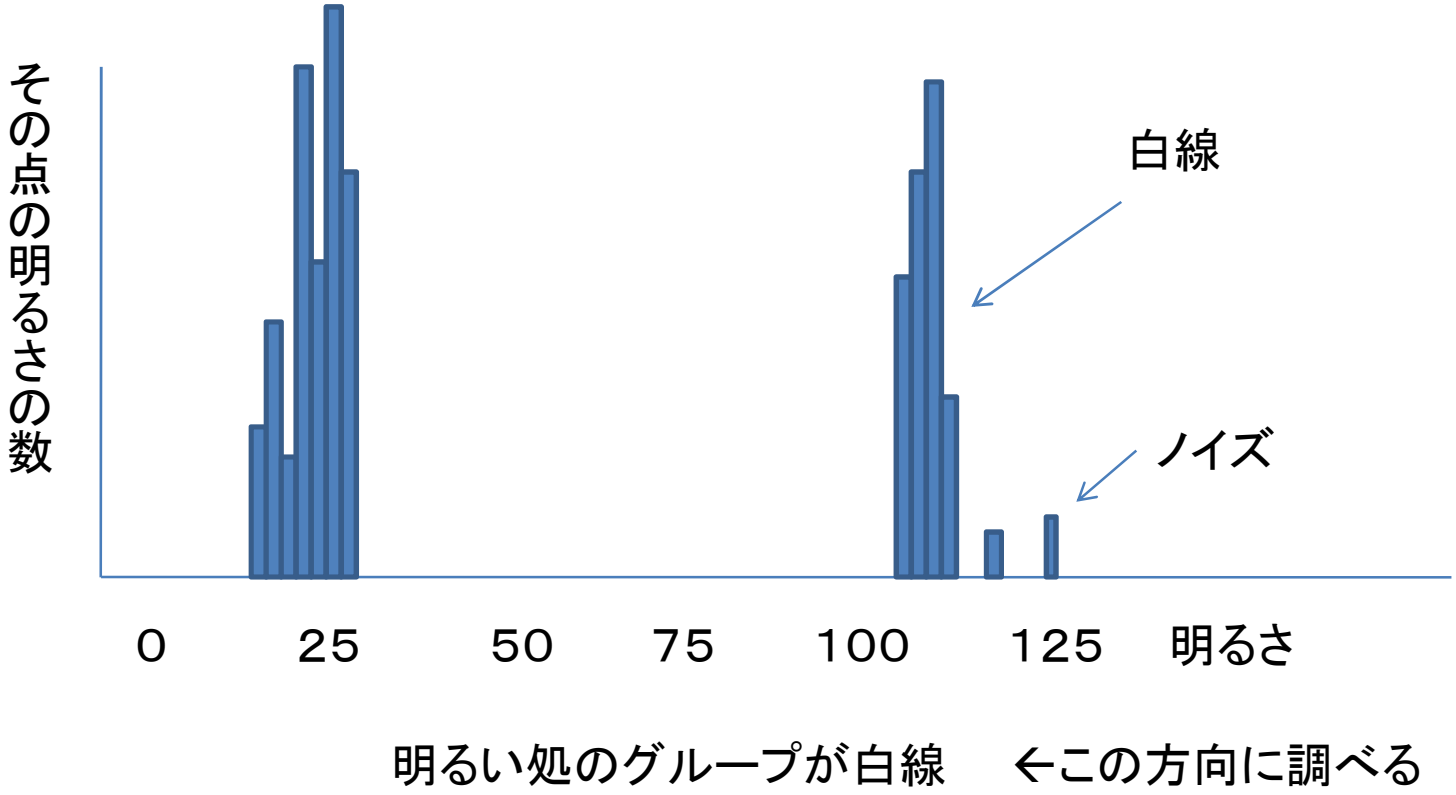


明るさの統計を取ってみる。

ヒストグラムから白線を検出する関数。

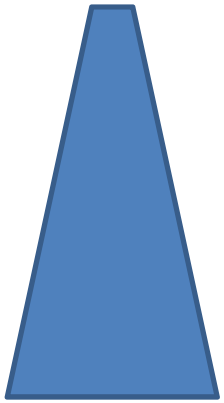
ヒストグラムから、ノイズと白線を分離する。

- int HistWhitLevel(void)



画像の補正

- ①画像は近くが大きく写り、遠くは小さくなります。
- ②画像は近くは間隔が広く写り、遠くは狭くなります。
- 車を走らすには、鳥が上空より見た様な、歪みのない、画面が理想です。
- ①の歪み補正は、各ラインで、違う倍率を掛算して、補正します。
- ②の歪み補正は、ラインの読み飛ばし数を変える事により行っています。



カメラで撮影すると、白線
はこの様に、写る。

アイちゃんの直線のトレース方法。

- マトリクスで得データは、8ライン分です。
- その中から、正しい直線を見つけなければ、なりません。
- まず最初に、一番近い直線のｾﾝﾀｰ付近が目指す直線と仮定します。
- その前のラインで、先ほどの直線検出位置に近い場所が、その直線の続きと予測します。
- それを、繰り返す事で、1本の直線を認識できます。

コンパイル環境の設定

１）１６MByteモードでコンパイルしています。

理由。

ヘッダファイル"3048f.h"が１６MByte用になっているためです。

（3048f.hは日立のホームページからダウンロードした物です。）

１MByteモードへの変更方法。

◎ヘッダファイル"3048f.h"の中のアドレスを記述している部分を２４Bitから２０Bitに変更してください。

即ち　0xffef10 --> 0x0fef10 の様に最上位のfを0に変更（全て）する。

◎スタートアップ（EyeChanStart.src）のスタックのアドレスも２０Bit表現に変更する。

２）構造体の扱い。

"3048f.h"は、Byte境界で合わせる様に作成されています。（Word境界ではありません）

"3048f.h"は、Bitフィルードの扱いが上位よりの割り当てで、作成されています。

３）インライン・アッセンブルを使用。

カメラに出す、タイミングの調整の為に、アッセンブラでnop命令を投入しています。

ツール・チェインのコンパイラタブの下にある、コンパイラオプションのフィールドに-code=asmcode を追加してください。（チェックボタン、ラジオボタンはありません。）

4)HEWのコンパイルで、コンパイル環境を変えた時、プリプロセッサ実行の前に、標準ライブラリ・ヘッダを読みに行きます。

その対策として、HEWのシステムに持っている
Hew¥Tools¥Renesas¥H8¥6_2_0¥includeにsysio.hを空で作成してください。

（ sysio.hはYellowの標準ヘッダです。）

- Yellowでコンパイルしたい時。

commonもホルダーにある、myType.hの以下の記述を変更してください。

Hewでコンパイル。

```
//#define __YH8Cx__    //コメントにする。
 #define __HEW__          //コメントを外す。
```

Yellowでコンパイル。

```
 #define __YH8Cx__      //コメントを外す。
 //#define __HEW__        //コメントにする。
```

今後の課題

- カメラで、現在のコース（材質の問題）を見ると、光線の関係で、ハレーション（光って、全体が白く写る）を起こして、コースの認識に失敗します。
  カメラの前に偏光フィルター（シートで売っている）を付けおると、水中の魚を水面の反射を防止して、写真を撮るのと一緒で、ある程度ハレーションを防止できる様です。
- カメラで見た画像が、どの様に見えているか、人間が、画像を直接見れると、良いのですが、データ量が多くて、128x128＝16KByteなので、H8/3048のRAMに入りきりません。
  RS232cでカメラから読んで、115Kbpsで送ると、1秒以上かかり、カメラの中のデータ（小さいコンデンサーに電荷として蓄えられている）が消えてしまいます。
  20KByte程度のRAM容量があるCPUで、読む必要があります。

- FPGA等を使用して、多量の情報を、精度よく、高速処理をする、ハードウエアの開発。

ご静聴ありがとうございました。