Introduction to Mechatronics

Programming a robot

Lecturer

Filippo Sanfilippo

Faculty of Maritime Technology and Operations
Aalesund University College, Norway

@fisa






SINAS


世界一美しい





Content of today's lecture

- Programming and software engineering in mechantronics
- The Robot Control Loop
- Communication
- Robot Programming Systems
  - Manual Programming
  - Automatic Programming
  - Software Architecture
- Programming in practice: Arduino
  - Programming a modular robot
  - Programming a delta robot

Schedule

- Overview and Introduction
- Evolutionary Robotics (GAs and Proximal vs. Distal Description)
- Neural Networks (Theoretical Introduction and examples)
- Arduino Seminar
- Prototyping and Mechanics Seminar

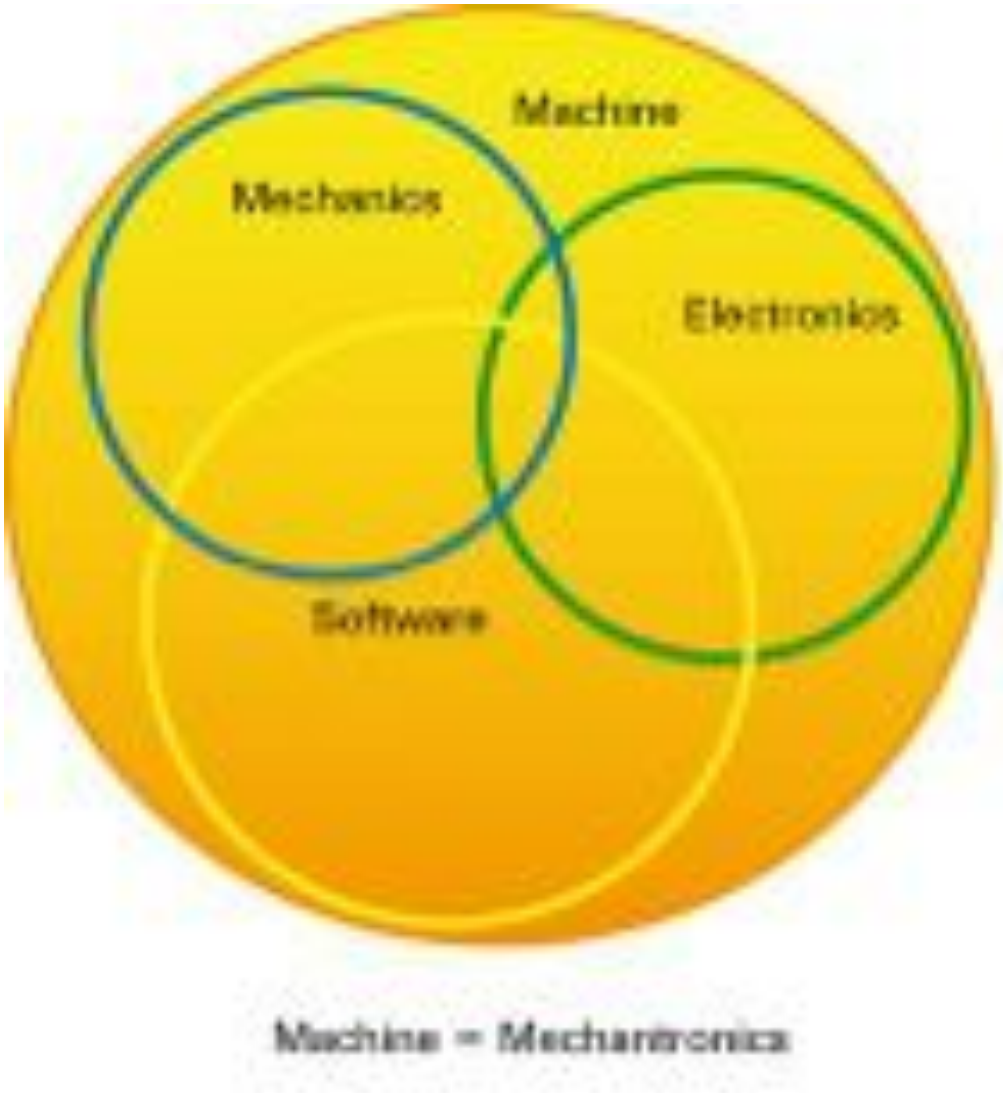
Lecture material

- **A survey of robot programming systems,** Biggs, G. and MacDonald, B., Proceedings of the Australasian conference on robotics and automation, pp. 1-3, 2003
- **Programming robots,** http://www.societyofrobots.com/programming_robot.shtml
- **Robot Programming Code**, http://www.cs.cmu.edu/~illah/ROBOCODE/index.html
- **Arduino**, http://www.arduino.cc/
- **Getting Started with Arduino,** Massimo Banzi, O'Reilly Media / Make, 2008
- **Making Things Talk, Practical Methods for Connecting Physical Objects**, Tom Igoe, O'Reilly Media / Make, 2007
- **The Mechatronics Handbook**, Robert H. Bishop CRC Press, 2002

Programming and software engineering in mechantronics

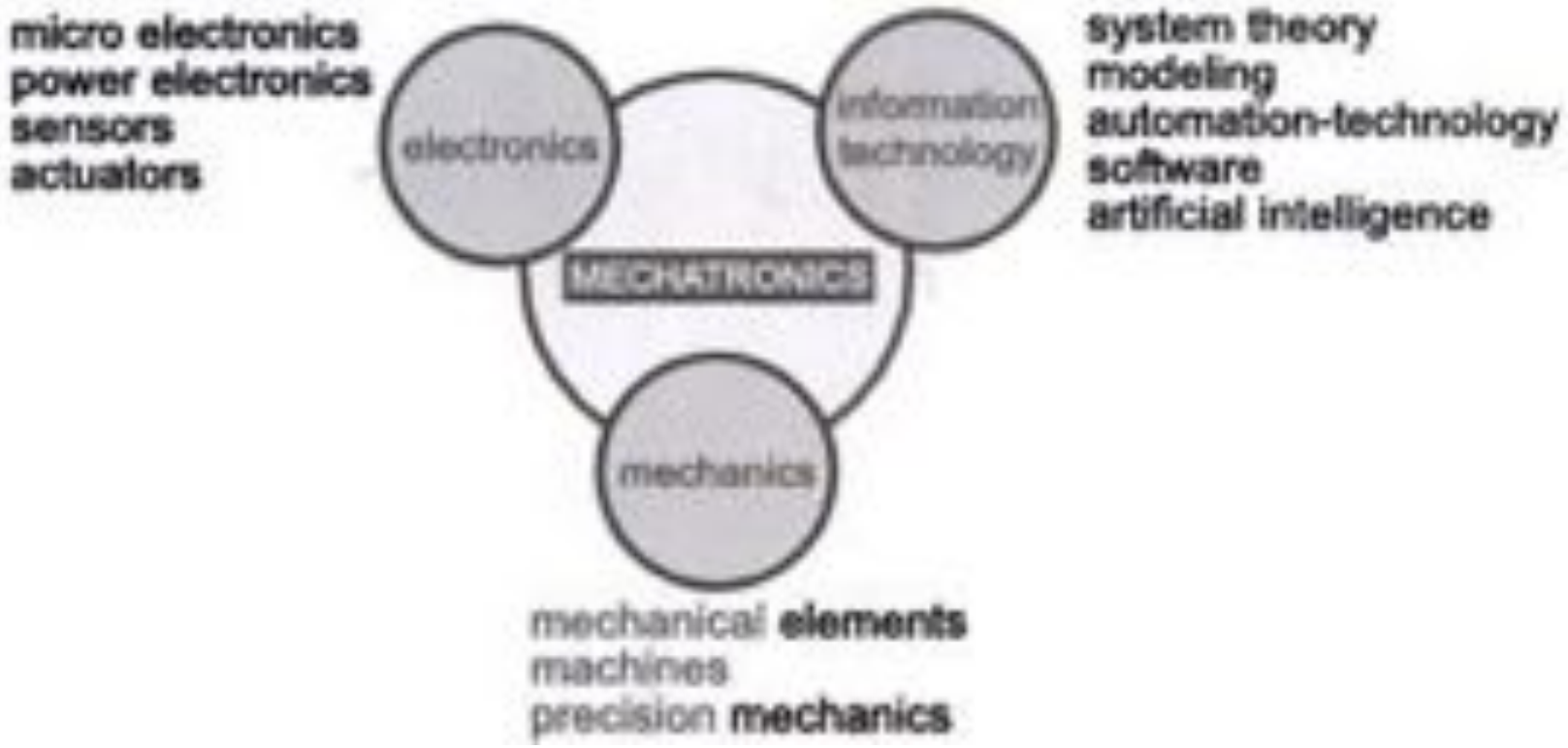
- Mechatronics is more than the sum of its parts
- Mechatronics as a way of thinking
- Mechanics, electronics and software are considered together and then simultaneously conceived, planned and implemented with interdisciplinary coordination.

-----> Software is the glue!



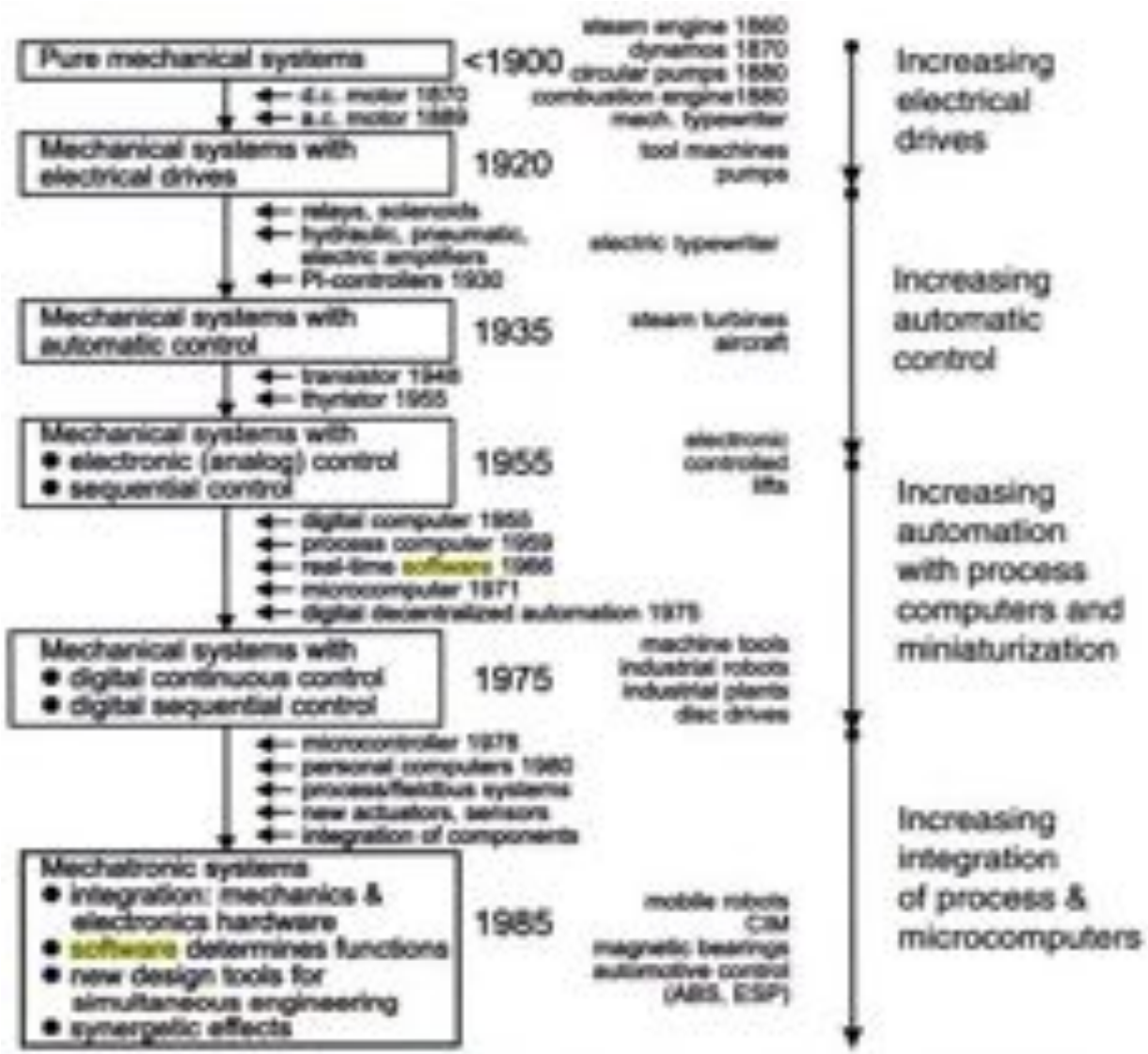
Programming and software engineering in mechantronics

- Software vs. Information Technology



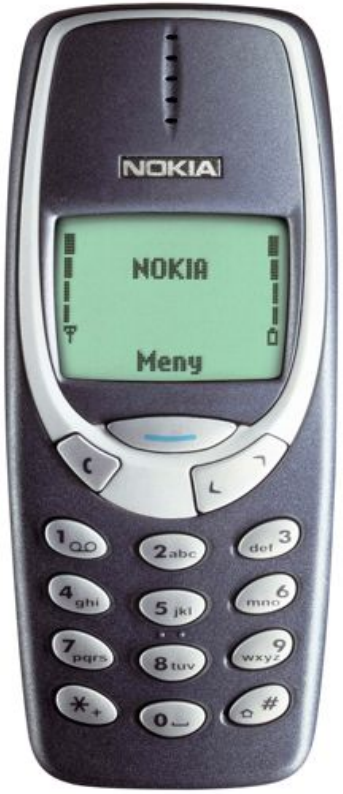
Programming and software engineering in mechantronics

- A little bit of history



Programming and software engineering in mechantronics

- Several overlaps between hardware and software
- Es. Mobile phone keyboard
- Analogic vs. Digital



Programming and software engineering in mechantronics

- Some properties of Conventional and Mechatronic Design Systems

Conventional Design	Mechatronic Design
Added components	**Integration of components (hardware)**
1 Bulky	Compact
2 Complex mechanisms	Simple mechanisms
3 Cable problems	Bus or wireless communication
4 Connected components	Autonomous units
Simple control	**Integration by information processing (software)**
5 Stiff construction	Elastic construction with damping by electronic feedback
6 Feedforward control, linear (analog) control	Programmable feedback (nonlinear) digital control
7 Precision through narrow tolerances	Precision through measurement and feedback control
8 Nonmeasurable quantities change arbitrarily	Control of nonmeasurable estimated quantities
9 Simple monitoring	Supervision with fault diagnosis
10 Fixed abilities	Learning abilities

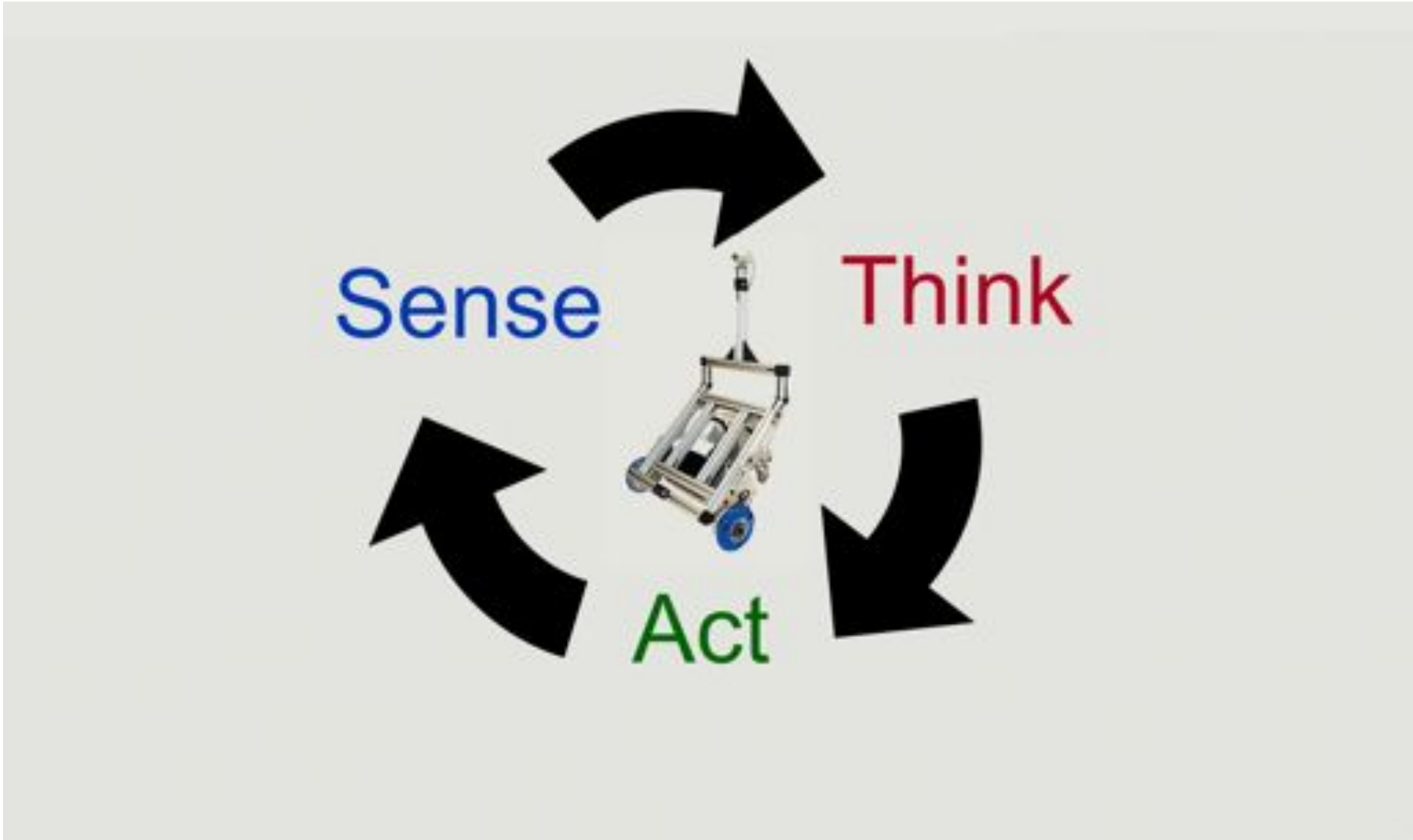
The Robot Control Loop

Open talk and sketches on the whiteboard

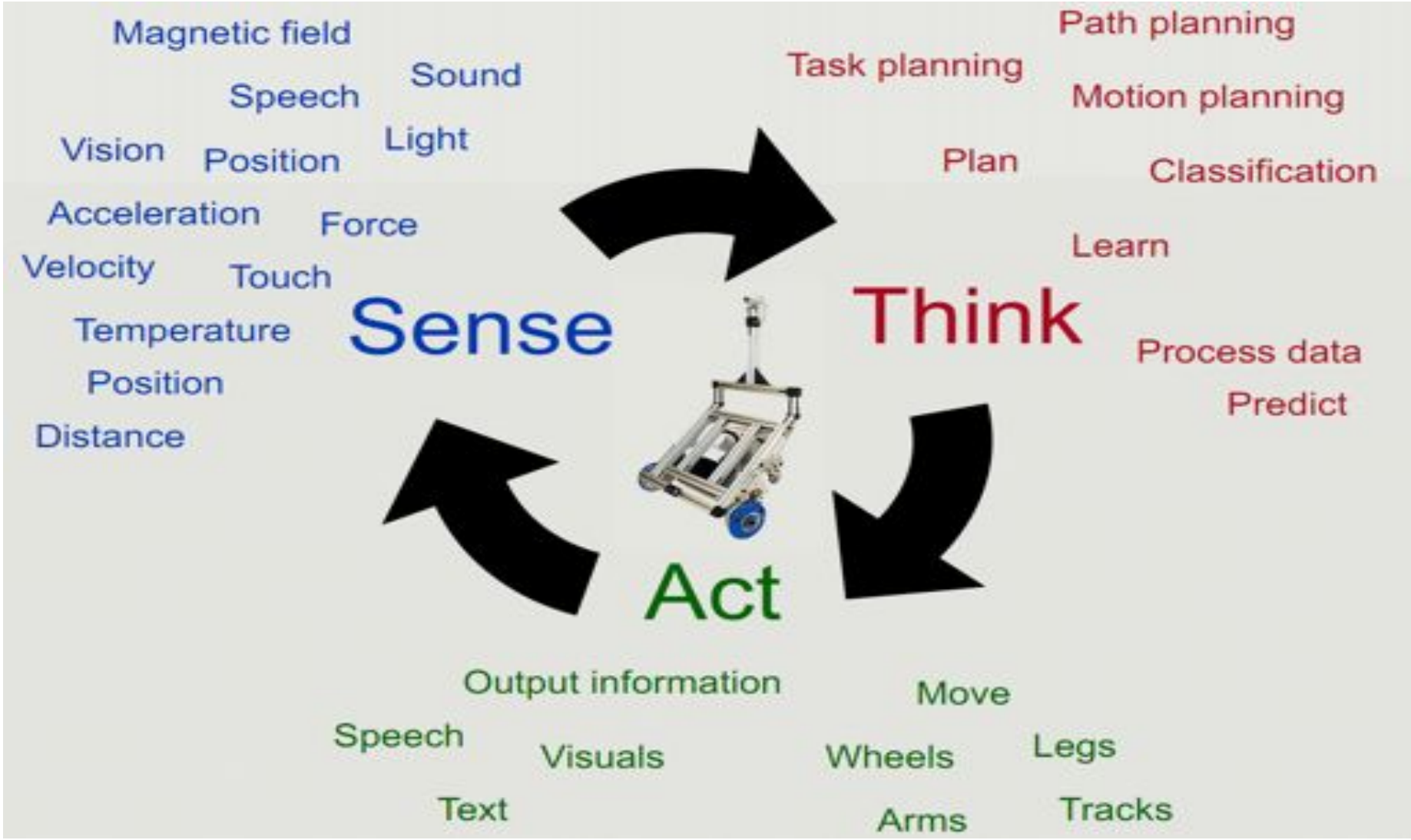
Take some notes!

"It's a blackboard. There is no spell-check."

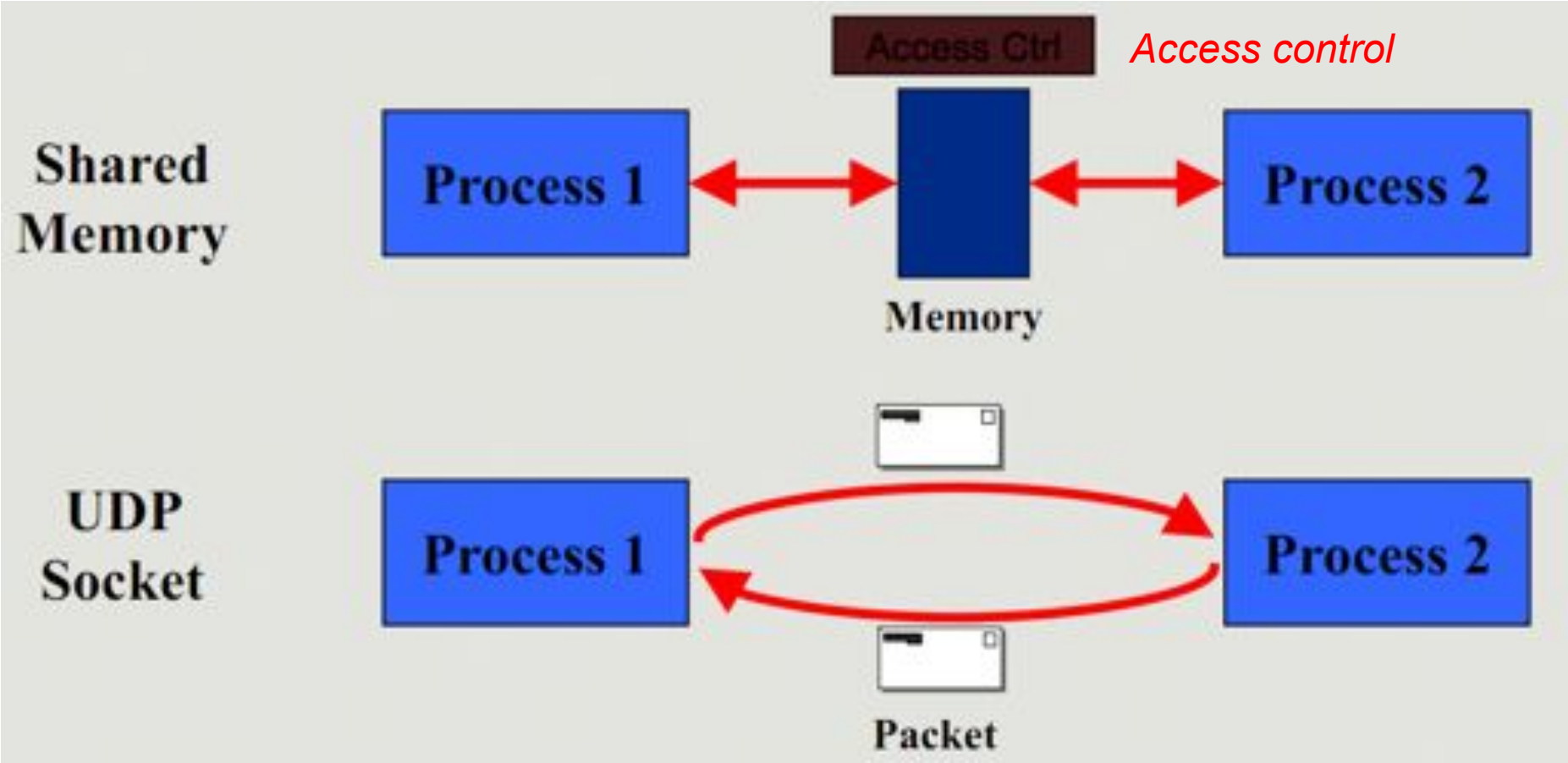
The Robot Control Loop



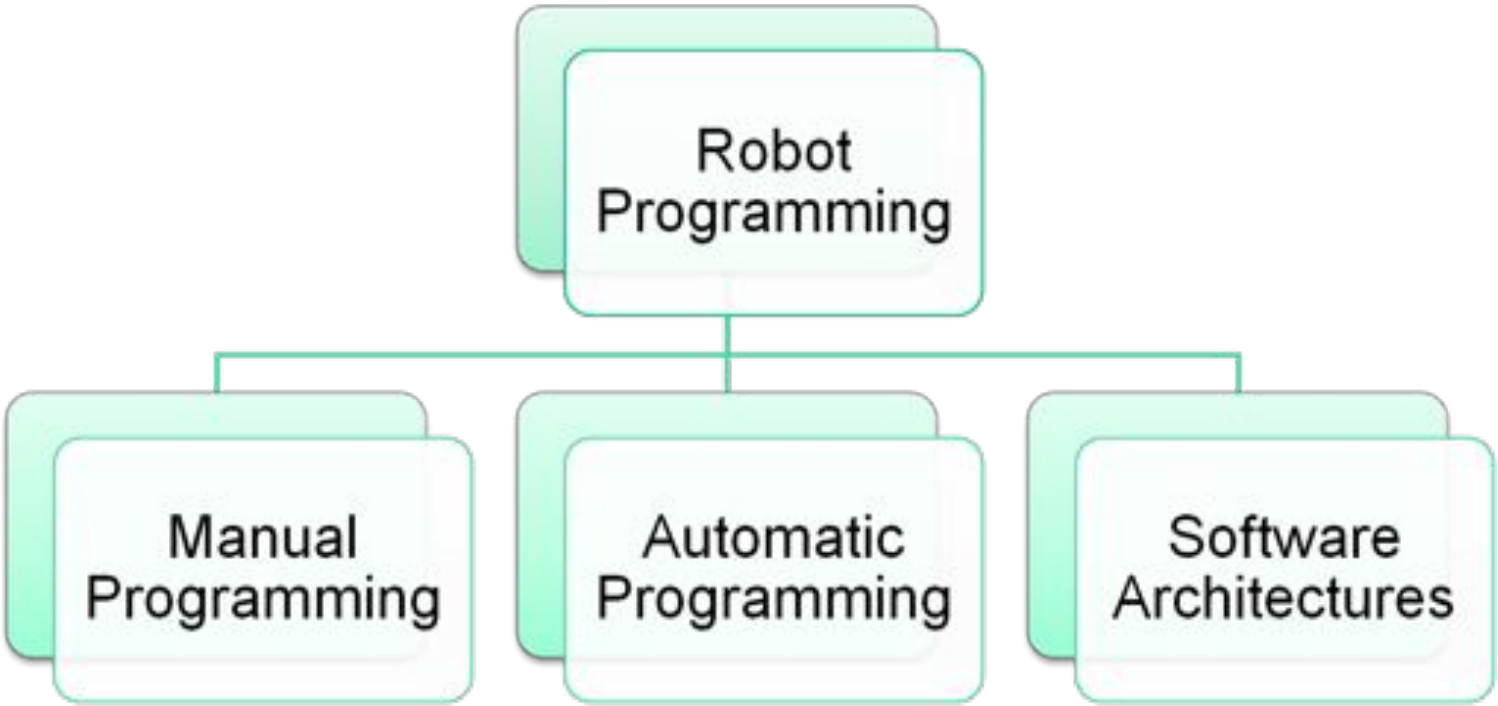
The Robot Control Loop



Communications



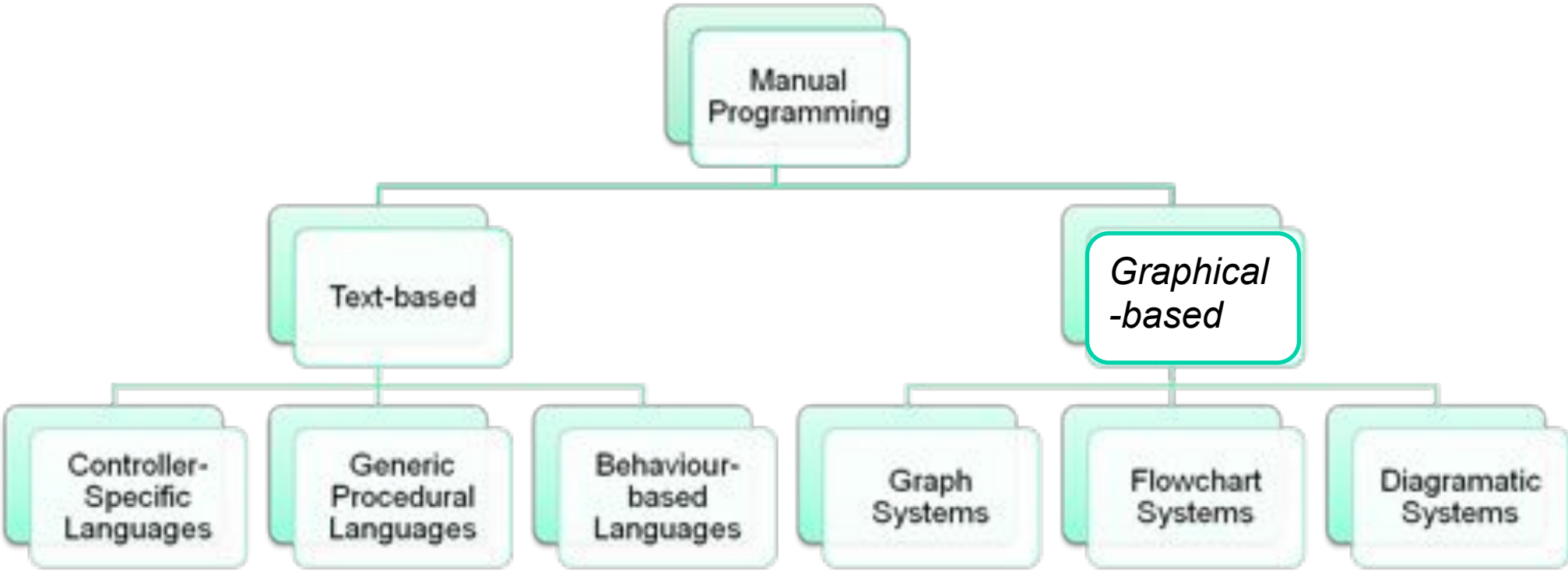
Robot Programming Systems



Manual Programming Systems

- Users of a manual programming systems must create the robot program by hand, which is typically performed without the robot. The finished program is loaded into the robot afterwards.
- These are often off-line programming systems, where a robot is not present while programming
- It is conceivable for manual programming to control a robot online, using for example an interpreted language, where there are no safety concerns

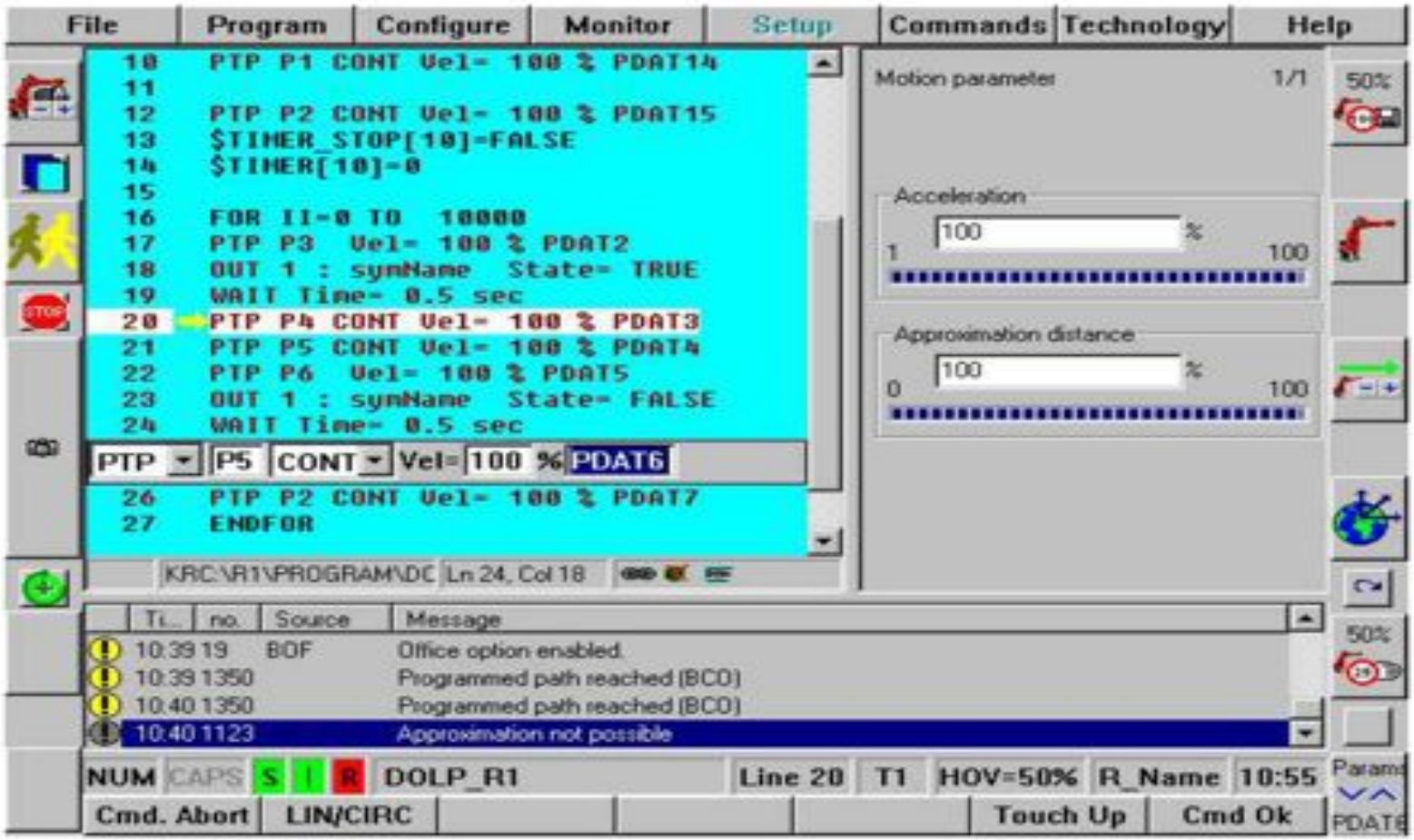
Manual Programming Systems



Controller-Specific Languages

- Controller-specific languages were the original method of controlling industrial robots, and are still the most common method today
- Every robot controller has some form of machine language, and there is usually a programming language to go with it that can be used to create programs for that robot
- These programming languages are usually very simple, with a BASIC-like syntax and simple commands for controlling the robot and program flow

Controller-Specific Languages



The KUKA programming environment and robot programming language

Controller-Specific Languages

- Drawbacks
  - Despite having existed for as long as industrial robots have been in use, controller-specific languages have seen only minor advances
  - The biggest problem is the lack of a universal standard between languages from different robot manufacturers
  - If a factory uses robots from many different manufacturers then they will need to either train their programmers for each one, or pay the manufacturer to develop the required programs for them
  - Commercial systems have concentrated their advances on overcoming these lacks by providing more advanced programming systems that remove the need for the programmer to write the robot code by hand

Generic Procedure Languages

- Generic languages provide an alternative to controller-specific languages for programming robots. "Generic" means a high-level multi-purpose language, for example C++ or JAVA
- particularly common in research environments, where generic languages are extended to meet the needs of the research project
- The choice of the base language varies, depending upon what the researchers are trying to achieve (for example, procedural or behavioral programming)

Generic Procedure Languages

- Advantages
  - The most common extension to a multi-purpose language is a robot abstraction, which is a set of classes, methods, or similar constructs that provides access to common robot functions in a simple way
  - They remove the need to handle low-level functionality such as setting output ports high, to turn on motors or translating raw sensor data
  - It might also provide higher-level abstractions, such as methods to make the robot move to a point using path planning

Generic Procedure Languages

- Drawbacks
  - these abstractions suffer from the same fault as controller-specific language. They are still specific to the robot they are designed for
- Possible improvements
  - many researches have developed their own robot abstraction systems
  - some examples:
    - Player/stage [Vaughan et al., 2003]
    - Motion Description Language [Kanayama and Wu, 2000]

Generic Procedure Languages

- Methodology to develop abstractions
  - it is interesting to note that a abstractions are commonly implemented using an object-oriented methodology
  - McKee et al. [2001] state that a rigorous object-oriented approach to robotics is important to clearly define robotic entities relationships. They describe the MARS model of object-oriented robots, and define robots comprised of "resources", which are then modeled as "modules". They draw clear parallels between object-oriented concepts such as inheritance, and the modules of a robot. Example: a tool on the end of an arm, simply by being on the end of an arm, inherits the ability to move.

Behaviour-based Languages

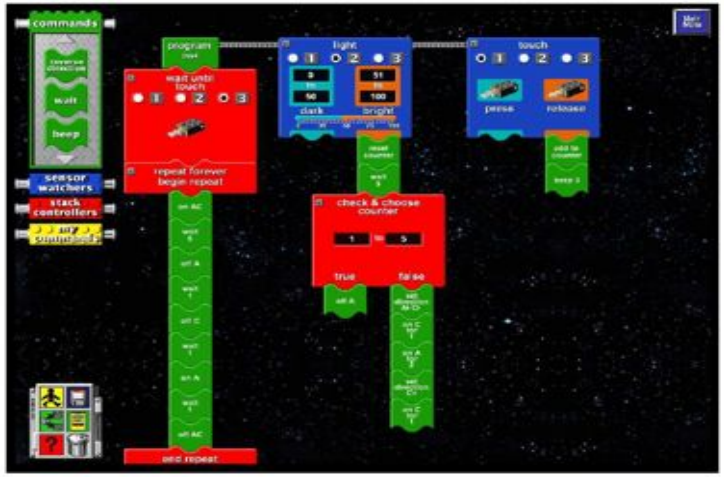
- They typically specify how the robot should react to different conditions, rather than providing a procedural description
- A behavioral system is more likely to be used by a robot developer than the end user. The developer would use it to define functionality that the end user would use to perform tasks
- Example: a set of behaviors would be to follow a wall from one point to another (a profile path is provided to the robot)

Graphical Systems

- Graphical (or icon-based) programming systems provide an alternative to text-based methods for manual programming
- they are a small step closer to automatic programming, as they provide a graphical medium for programming
- they require manual input to specify actions and program flow.
- Graphical systems typically use a graph, flow-chart or diagram view of the robot system

Graphical Systems

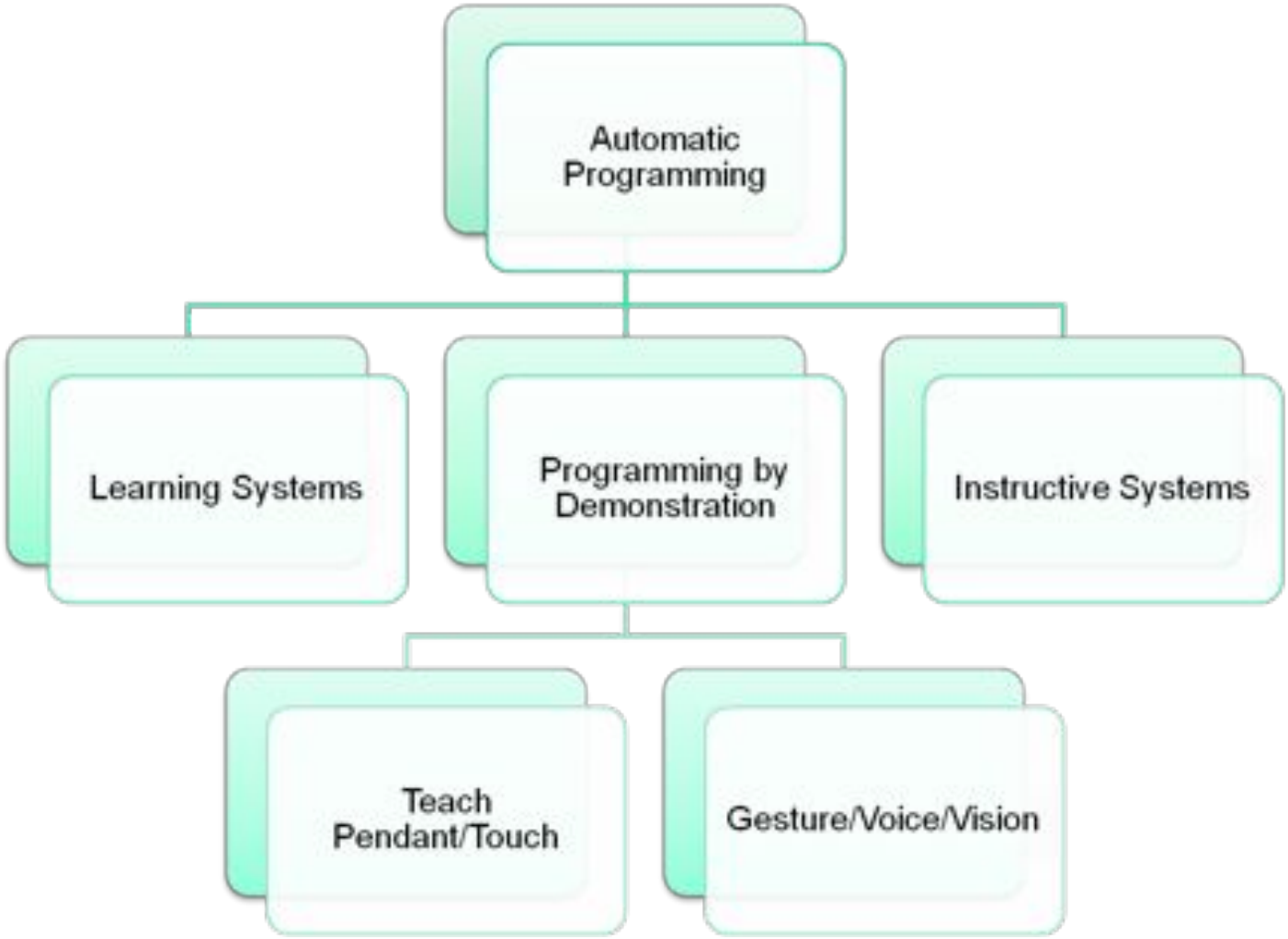
- One advantage of graphical systems is their ease of use, which is achieved at the cost of text-based programming's flexibility
- They are typically used for robotic applications rather than system programming
- Perhaps the most successful graphical system using the flow-chart approach is employed by the Lego Mindstorms robotics kit [Lego, 2003]



Automatic Programming Systems

- Automatic programming systems provide little or no direct control over the program code the robot will run
- robot code is generated from information entered into the system in a variety of indirect ways
- often a robot system must be running while automatic programming is performed, and these systems have been referred to as "online" programming systems
- automatic programming may also be performed on simulated or virtual robots

Automatic Programming Systems



Learning Systems

- Learning systems create a program by inductive inference from user provided examples and self-exploration by the robot
- Initially the robot watches as the task is performed. Successively the robot attempts to perform the task on its own
- Examples include a hierarchy of neural networks developed for learning the motion of a human arm in 3D [Billard and Schaal, 2001], a robot that can learn simple behaviours and chain these together to form larger behaviours [Weng and Zhang, 2002], ...

Programming by Demonstration

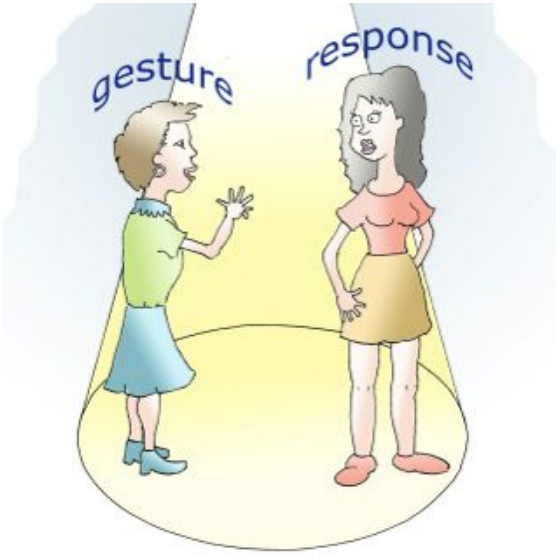
- This is the most common method of automatic programming
- PbD systems may use touch/pendants for the demonstration, or they may use other, more natural communication methods such as gestures and voice

Programming by Demonstration

- A traditional PbD system uses a teach-pendant to demonstrate the movements the robot should perform
- The demonstrator performs the task (for example, an assembly task) using the teach pendant
- The position of the pendant is recorded and the results used to generate a robot program that will move the robot arm through the same motions
- Alternatively, the demonstrator may move the robot arm through the required motions either physically or using a controller. In this case, the joint positions are recorded and used to generate the robot program.

Instructive Systems

- Instructive systems are given a sequence of instructions, usually in real-time.
- The technique is best suited for commanding robots to carry out tasks that they have already been trained or programmed to perform
- it could be considered the highest level of programming. Typically, gesture recognition or voice recognition is used



Programming in practice

- Robots have become significantly more powerful and intelligent over the last decade
- Robots will more often be used by people with minimal technical skills
- Free and Open Source Software (FOSS) and Open source hardware (OSHW) can bee seen as very promising means for the technological progress

Arduino: introduction

- What is Arduino?
  - It's an open-source physical computing platform based on a simple microcontroller board (Atmel's ATMEGA8 and ATMEGA168 microcontrollers), and a development environment for writing software for the board.
  - It can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs.
  - Arduino projects can be stand-alone, or they can be communicate with software running on a computer (e.g. Flash, Processing, MaxMSP.) The boards can be assembled by hand or purchased preassembled; the open-source IDE can be downloaded for free.
  - Arduino The Documentary: http://vimeo.com/18539129

Arduino: introduction

- Why Arduino?
  - Arduino simplifies the process of working with microcontrollers. It also offers the following advantages:
  - Inexpensive compared to other microcontroller platforms.
  - Cross-platform - Windows, Macintosh OSX, and Linux.
  - Simple, clear programming environment
  - Open source and extensible software - The Arduino software and is published as open source tools, available for extension by experienced programmers.
  - Open source and extensible hardware - The plans for the modules are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it.

Arduino: getting started

- Installation
  - Step-by-step instructions for setting up the Arduino software and connecting it to an Arduino Uno, Mega2560, Duemilanove, Mega, or Diecimila.
  - Windows
  - Mac OS X
  - Linux (on the playground wiki)

Environment: Description of the Arduino development environment.

Troubleshooting: Advice on what to do if things don't work.

Arduino: hardware

- Official Arduino boards

Arduino Uno

Arduino LilyPad

Arduino Pro

Arduino Nano

Arduino Mini

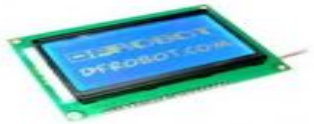
Arduino: hardware

- Official Arduino Shields and Modules

Arduino Ethernet Shield

Arduino Wireless Shield

GPRS Quadband Module



LCD Module

GPS Module

Touch Sensor

Accelerometer

IR Receiver

Arduino: learning

- Sketch: Arduino uses for a program. It's the unit of code that is uploaded to and run on an Arduino board.
  - **Comments**

    Everything between the /* and */ is ignored by the Arduino when it runs the sketch. There's another style for short, single-line comments. These start with // and continue to the end of the line.
  - **Variables**

    A variable is a place for storing a piece of data. It has a name, a type, and a value. There's more information in the Variables tutorial.

NOTE: refear to the Blink example.

Arduino: learning

Sketch

- **Functions**

  A function (otherwise known as a procedure or sub-routine) is a named piece of code that can be used from elsewhere in a sketch. You can call a function that's already been defined (either in your sketch or as part of the Arduino language).

- **setup() and loop()**

  There are two special functions that are a part of every Arduino sketch: setup() and loop(). The setup() is called once, when the sketch starts. It's a good place to do setup tasks like setting pin modes or initializing libraries. The loop() function is called over and over and is heart of most sketches. You need to include both functions in your sketch, even if you don't need them for anything.

Arduino: learning

Microcontrollers

- **Digital Pins**

  The pins on the Arduino can be configured as either inputs or outputs. Arduino (Atmega) pins default to inputs, so they don't need to be explicitly declared as inputs with pinMode().

- **Analog Input Pins**

  The Atmega controllers used for the Arduino contain an onboard 6 channel analog-to-digital (A/D) converter. While the main function of the analog pins for most Arduino users is to read analog sensors, the analog pins also have all the functionality of general purpose input/ output (GPIO) pins (the same as digital pins 0 – 13). The analog pins can be used identically to the digital pins, using the aliases A0 (for analog input 0), A1, etc.

Arduino: learning

Microcontrollers

- **PWM**

  Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. The Fading example demonstrates the use of analog output (PWM) to fade an LED. It is available in the File->Sketchbook->Examples->Analog menu of the Arduino software.

Arduino: learning

Microcontrollers

– **Memory**

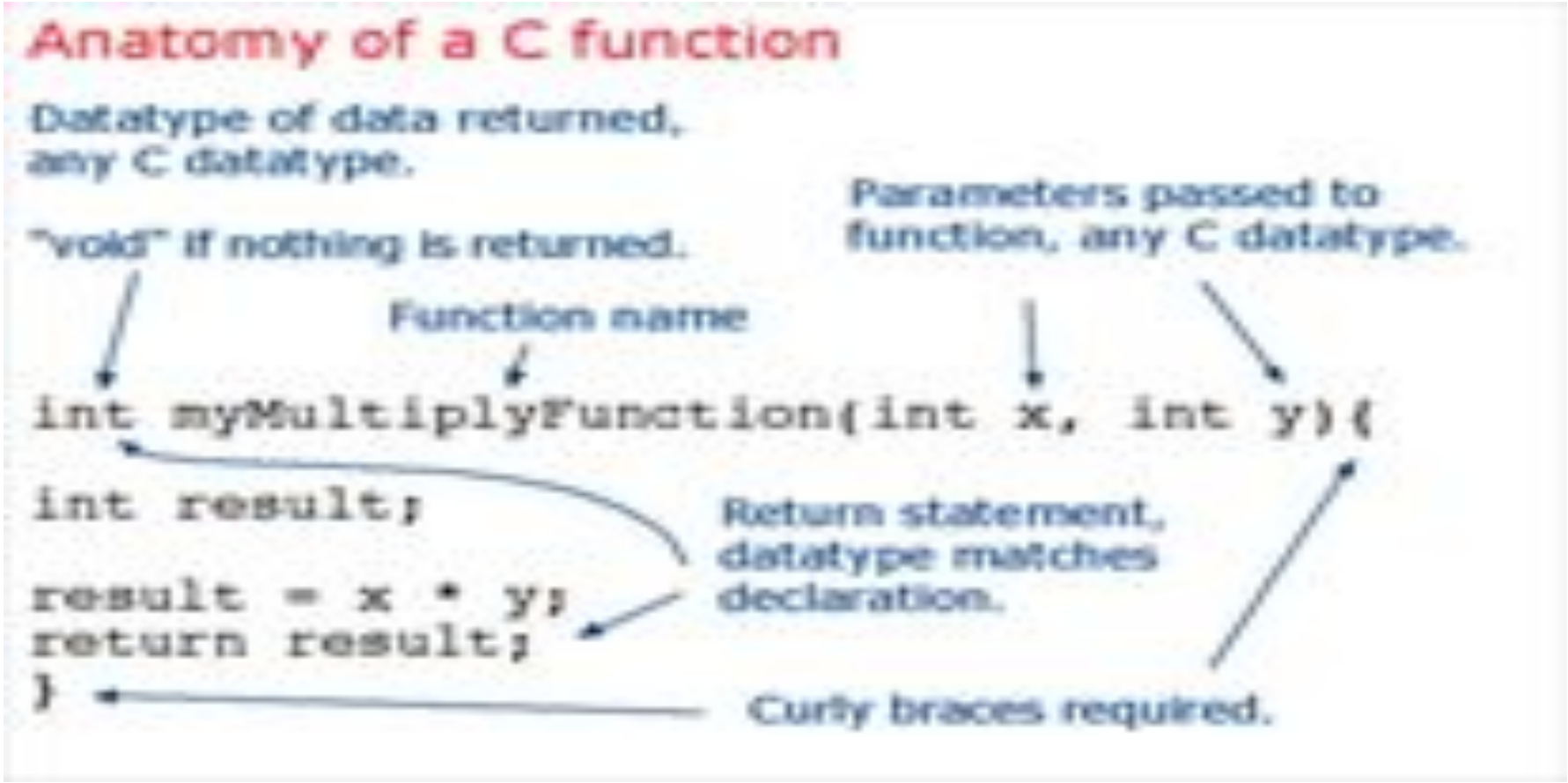
There are three pools of memory in the microcontroller used on Arduino boards (ATmega168):

- Flash memory (program space), is where the Arduino sketch is stored.
- SRAM (static random access memory) is where the sketch creates and manipulates variables when it runs.
- EEPROM is memory space that programmers can use to store long-term information.

Arduino: learning

Programming Technique

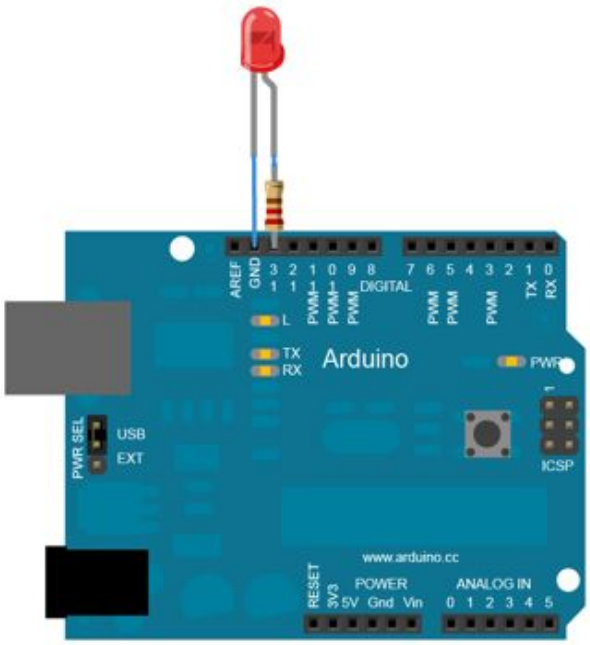
– **Functions**



Arduino: learning by doing

Examples

– Blink



check out:

- Blink
- Blink without delay

Circuit

To build the circuit, attach a 220-ohm resistor to pin 13. Then attach the long leg of an LED (the positive leg, called the anode) to the resistor. Attach the short leg (the negative leg, called the cathode) to ground. Then plug your Arduino board into your computer.

In the program below, the first thing you do is to initialize pin 13 as an output pin with the line

pinMode(13, OUTPUT);

In the main loop, you turn the LED on with the line:

digitalWrite(13, HIGH);

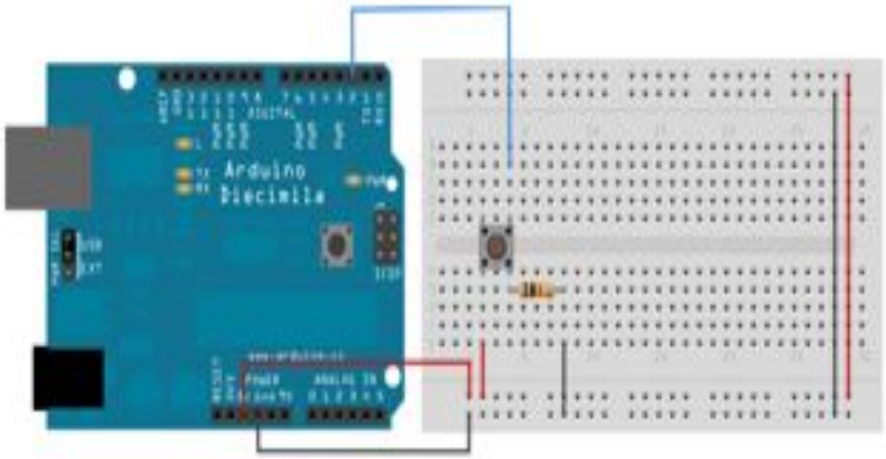
This supplies 5 volts to pin 13. That creates a voltage difference across the pins of the LED, and lights it up. Then you turn it off with the line:

digitalWrite(13, LOW);

Arduino: learning by doing

Examples

– Button



check out:

• Button

Circuit

Connect three wires to the Arduino board. The first two, red and black, connect to the two long vertical rows on the side of the breadboard to provide access to the 5 volt supply and ground. The third wire goes from digital pin 2 to one leg of the pushbutton. That same leg of the button connects through a pull-down resistor (here 10 KOhms) to ground. The other leg of the button connects to the 5 volt supply.

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}
...
  buttonState = digitalRead(buttonPin);
  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  }
  else {
    digitalWrite(ledPin, LOW);
  }
```

Arduino: learning by doing

Examples

– Analog Input

Circuit

Connect three wires to the Arduino board. The first goes to ground from one of the outer pins of the potentiometer. The second goes from 5 volts to the other outer pin of the potentiometer. The third goes from analog input 0 to the middle pin of the potentiometer.

```
void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

check out:

- Analog Input

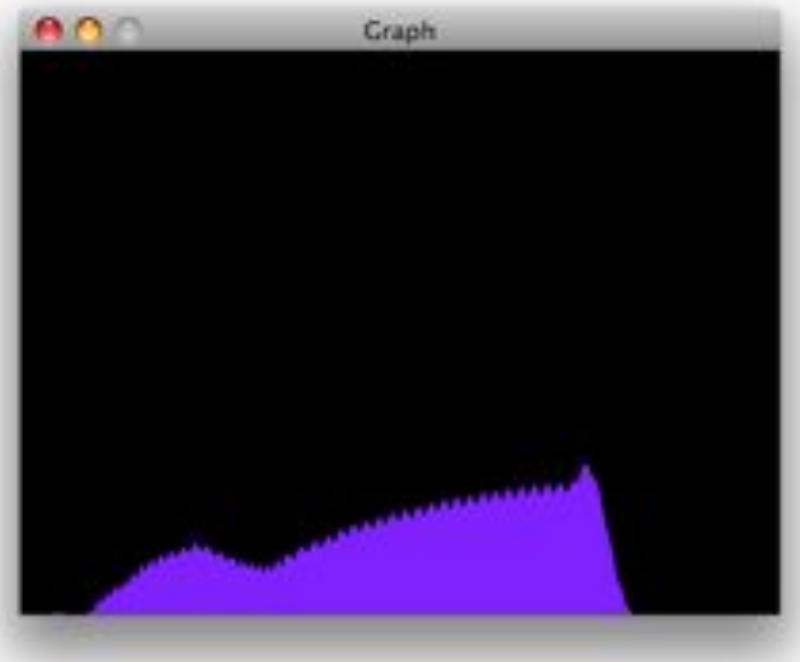
Arduino: learning by doing

Examples

– Graph

Circuit

This example shows you how to send a byte of data from the Arduino to a personal computer and graph the result. This is called serial communication. You can use the Arduino serial monitor to view the sent data, or it can be read by Processing.



check out:

- Graph
- Processing

Arduino: learning by doing

Examples

- **Accelerometer**

check out:

- Accelerometer

Circuit

You'll use three of the analog input pins as digital I/O pins, for power and ground to the accelerometer, and for the self-test pin. You'll use the other three analog inputs to read the acclerometer's analog outputs.

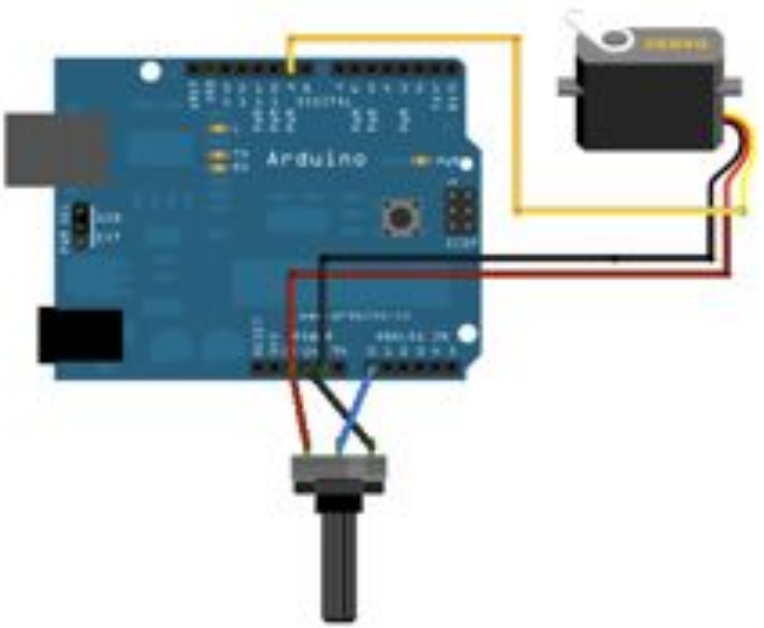
Reads an Analog Devices ADXL3xx accelerometer and communicates the acceleration to the computer.

```
void loop()
{
  Serial.print(analogRead(xpin));
  Serial.print("\t");
  Serial.print(analogRead(ypin));
  Serial.print("\t");
  Serial.print(analogRead(zpin));
  Serial.println();
  // delay before next reading:
  delay(100);
}
```

Arduino: learning by doing

Examples

– Knob



Circuit

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow or orange and should be connected to pin 9 on the Arduino board.

The potentiometer should be wired so that its two outer pins are connected to power (+5V) and ground, and its middle pin is connected to analog input 0 on the Arduino.

```
void loop()
{
  val = analogRead(potpin);
  val = map(val, 0, 1023, 0, 179);
  myservo.write(val);
  delay(15);
}
```

check out:

- Knob

Parallel robots

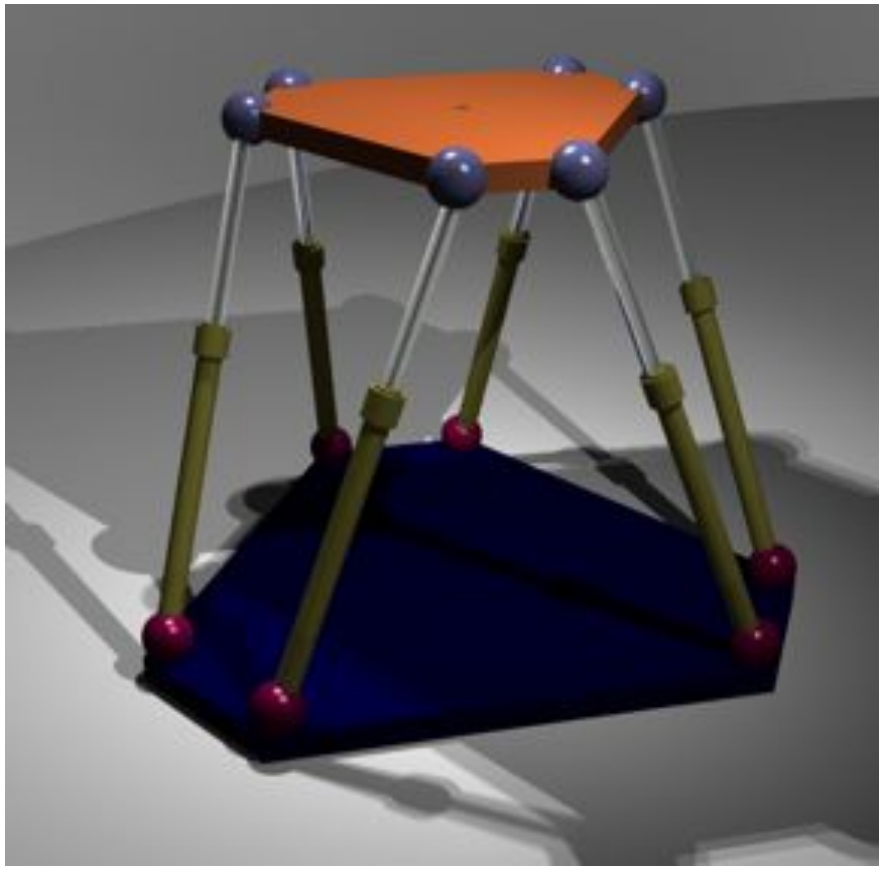
A parallel robot is a closed loop chain, whereas a serial robot is an open loop chain (a hybrid mechanism is one with both closed and open chains).

Major industrial applications of these devices are:

- flight simulators
- automobile simulators
- in work processes
- photonics / optical fiber alignment

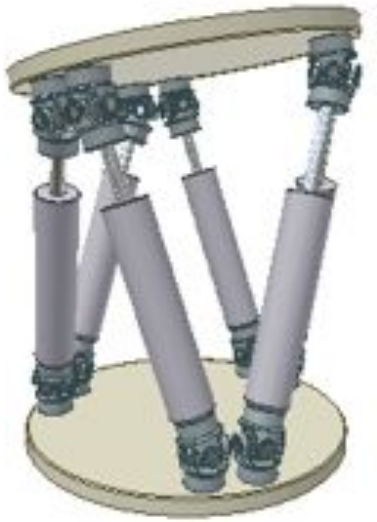
They also become more popular:

- in high speed, high-accuracy positioning with limited workspace, such as in assembly of PCBs
- as micro manipulators mounted on the end effector of larger but slower serial manipulators
- as high speed/high-precision milling machines

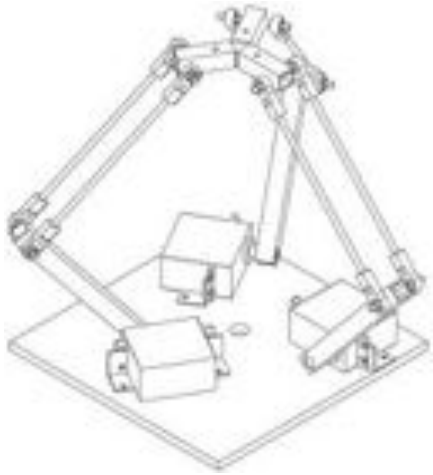


Parallel robots

Two examples of popular parallel robots are the Stewart platform and the Delta robot.



Stewart platform



Delta robot

Delta robot

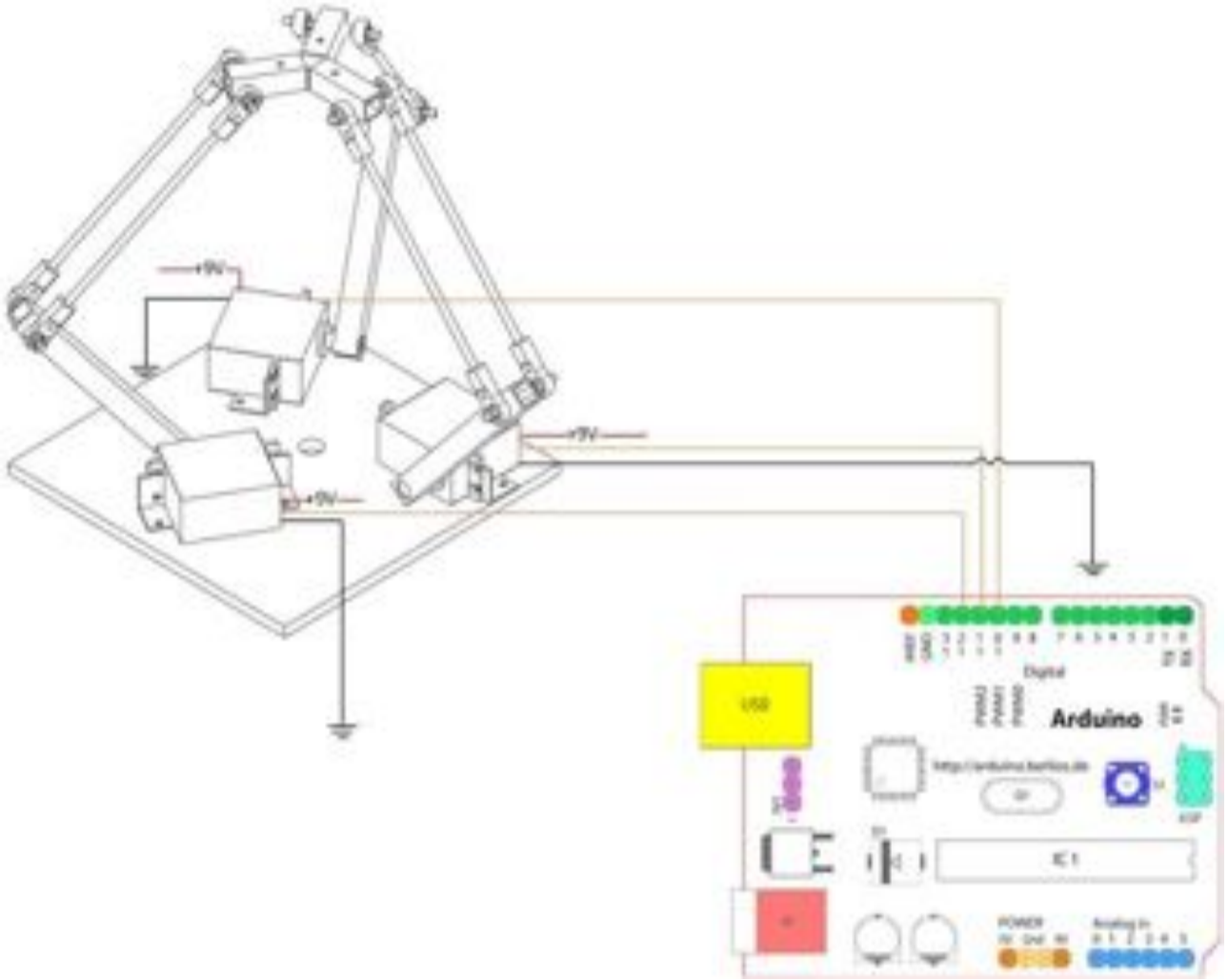
A delta robot is a type of parallel robot. It consists of three arms connected to universal joints at the base. The key design feature is the use of parallelograms in the arms, which maintains the orientation of the end effector.

The key concept of the Delta robot is the use of parallelograms. These parallelograms restrict the movement of the end platform to pure translation (only movement in the X, Y or Z direction).

Because the actuators are all located in the base, and the arms can be made of a light composite material the moving parts of the Delta robot have a small inertia. This allows for very high accelerations.

Check out:
Kinematics Model

Programming a delta robot



The robot can be controlled by using:

- keyboard
- joystick
- mouse
- accelerometer
- ...

Check out:
A parallel delta robotcontrolled via iPhone accelerometer

Homework 01: Servo as input device

Reading the voltage from the center pin of the a servo's potentiometer so that it can be used as an input as well as an output device.
Useful for:

- Collision detection on unpowered arms (or even poor man's torque detection by measuring the difference between what you requested and the actual position, assuming the servo doesn't break);
- Physical keyframing; you move the arms of the thing you are animating manually, then press a button to record that position, repeat a number of times and then have the computer play it back;
- Haptic feedback, if you can control the servo fast enough (doubtful, but worth a try).

Check out:
Servo as input device
Robot Arm Remote Control Interface

Homework 01: Servo as input device

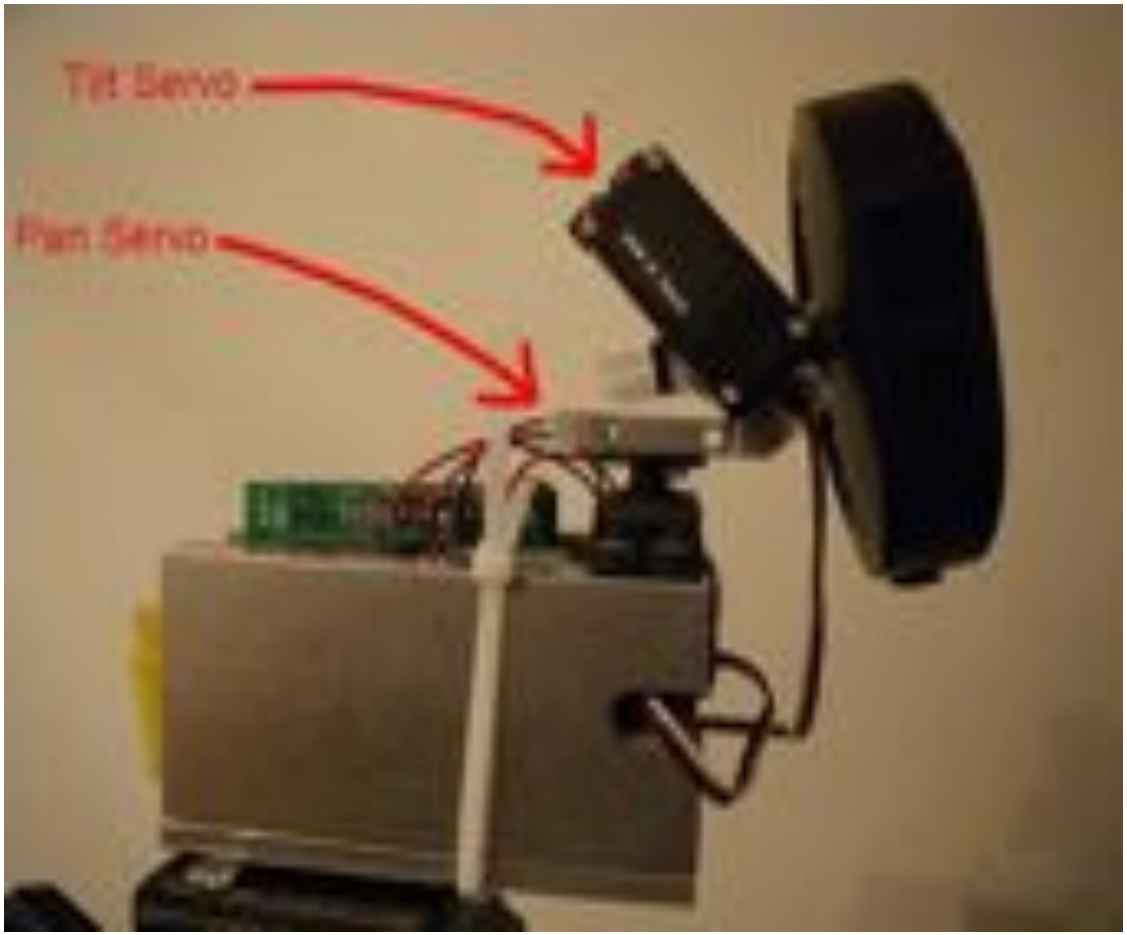
http://www.instructables.com/id/Servo-Feedback-Hack-free/

http://www.cibomahto.com/2008/02/thing-a-day-day-9-servo-as-input-device/

http://letsmakerobots.com/content/sensing-your-servos-position

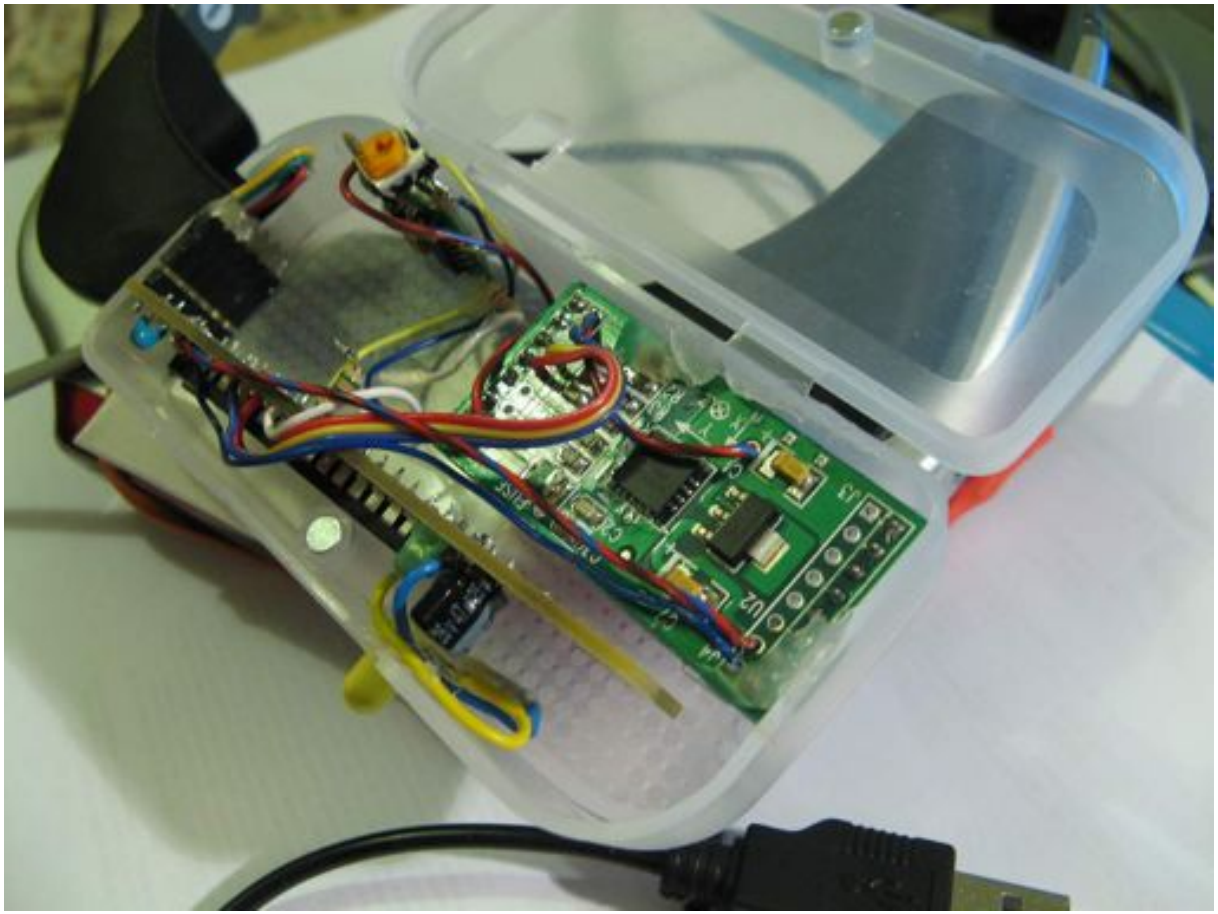
Check out:
Servo as input device
Robot Arm Remote Control Interface

Homework 02: Web Based Servo Control



Check out:
Web Based Servo Control

Homework 03: Arduino 3DOF Head Tracker

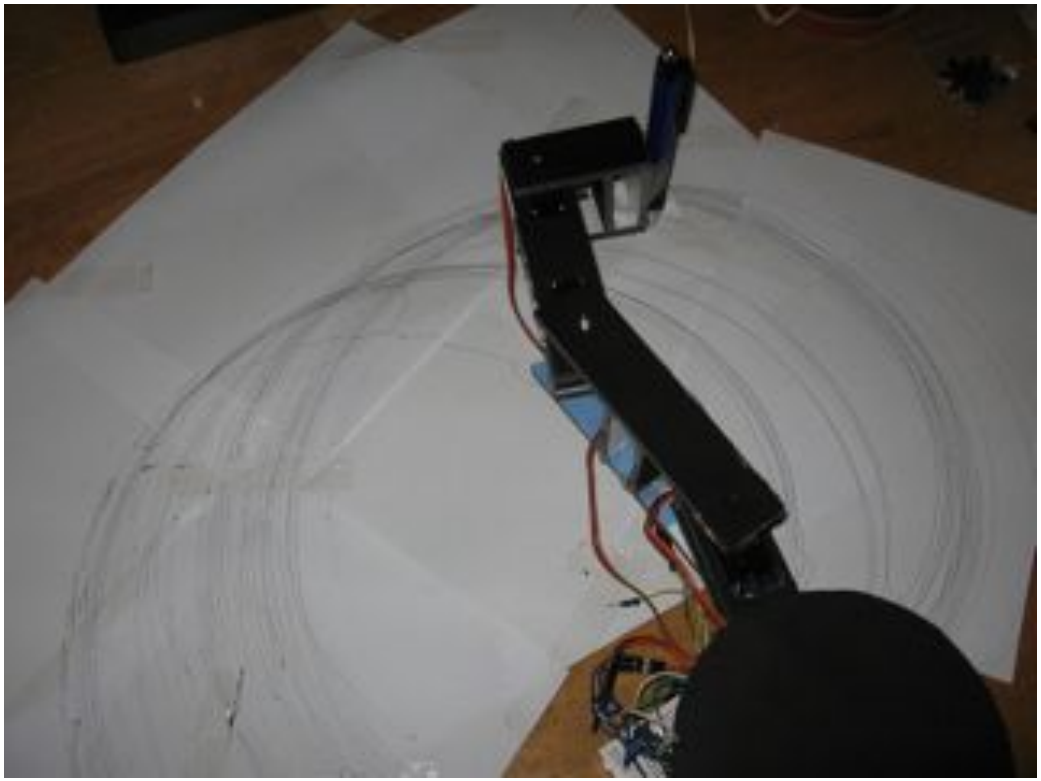


Check out:
Arduino 3DOF Head Tracker

Homework 04: Pong with the Arduino

Check out:
Pong with the Arduino

Homework 05: Arduino Robotic Arm



Build a basic Arduino robotic arm using 3 servos.

Check out:
Arduino robotic arm

Exercises

- Binary LEDs Clock
- Temperature monitor (LCD and some LEDs)
- Finger controlled servo (using a flex sensor)
- Relaxing light (photo resistor and LEDs)

Other material (AI and ANN)

- http://filipposanfilippo.inspitivity.com/presentations/evolutionary-robotics-sanfilippo.pdf
- http://arxiv.org/pdf/cs/0308031.pdf

Thanks for your attention!

Any questions?